Vom Bauwerksinformationsmodell zur Terminplanung

Ein Modell zur Generierung von Bauablaufplänen

Dissertation zur Erlangung des akademischen Grades

Doktor - Ingenieur

an der Fakultät Bauingenieurwesen
der
Bauhaus-Universität Weimar

vorgelegt von

Eike Tauscher
geboren am 30. November 1974
in Stollberg/Erzg.

Gutachter: 1. Prof. Dr.-Ing. Karl Beucke, Bauhaus-Universität Weimar

2. Prof. Dr.-Ing. Markus König, Ruhr-Universität Bochum

3. Prof. Dr.-Ing. Frank Petzold, Technische Universität München

Eingereicht am: 9. Februar 2011

Tag der Disputation: 19. August 2011

Vorwort

Die vorliegende Arbeit entstand in der Zeit von September 2006 bis Februar 2011 während meiner Tätigkeit als wissenschaftlicher Mitarbeiter an der Professur Informatik im Bauwesen an der Bauhaus-Universität Weimar. Während dieser Zeit bearbeitete ich das Thema im Rahmen des von der Deutschen Forschungsgemeinschaft (DFG) geförderten Projekts: „Modellierung von Ausführungsvarianten in der Bauprozessplanung“.
Ich möchte mich an dieser Stelle bei all jenen Personen bedanken, die zum Gelingen dieser Arbeit beigetragen haben.
Mein besonderer Dank gilt meinem Mentor Prof. Karl Beucke. Er hat mir nicht nur die für die Anfertigung der Arbeit nötigen Freiräume geschaffen sondern stand mir stets auch mit seinen wissenschaftlichen und organisatorischen Ratschlägen zur Seite. Weiterhin danke ich Prof. Markus König für die Erstellung des Gutachtens und darüber hinaus für die vielen zielführenden Diskussionen während der Bearbeitung des Forschungsprojekts. Er war während und auch nach dieser Zeit stets als Ansprechpartner für mich verfügbar. Bei Prof. Frank Petzold möchte ich mich für die Begutachtung dieser Arbeit bedanken.
Meiner Projektpartnerin Eva Mikulakova danke ich für die Zusammenarbeit und die vielen spannenden Diskussionen. Ebenso danke ich in diesem Zusammenhang Michael Theiler.
Für die freundschaftliche Arbeitsatmosphäre bedanke ich mich bei allen Kollegen am Lehrstuhl Informatik im Bauwesen. Besonders Jens-Uwe Wagner und Christa Diez danke ich für die vielen aufmunternden Worte und ihre uneingeschränkte Hilfe während meiner Zeit am Lehrstuhl.
Nicht zuletzt danke ich meiner Familie und meinen Freunden für die persönliche Unterstützung während der vergangenen Jahre. Aus diesem Personenkreis danke ich ganz herzlich Sonja Mathes für das schnelle Korrekturlesen dieser Arbeit.
Mein besonders herzlicher Dank gilt Claudia Grieser, die nicht nur meine körperliche und zum Teil geistige Abwesenheit respektiert hat. Sie hat mir immer - mit für sie großen persönlichen Einschränkungen - den Rücken freigehalten. Ohne ihre Unterstützung wäre die Anfertigung dieser Arbeit nicht möglich gewesen.

Weimar, im August 2011

Eike Tauscher

Für Claudia, Liv und Ian.

Kurzfassung

Die effiziente und zielgerichtete Ausführung von Bauvorhaben wird in hohem Maße von der zugrunde liegenden Bauablaufplanung beeinflusst. Dabei ist unter Verwendung herkömmlicher Methoden und Modelle die Planung des Bauablaufs ein zumeist aufwändiger und fehlerträchtiger Prozess.
Am Ende der gegenwärtig üblichen Vorgehensweise für die Planung eines Bauablaufs erfolgt lediglich die Dokumentation des Endergebnisses. Mögliche Ablaufalternativen, die im Verlauf der Planung betrachtet wurden, sind im resultierenden Bauablaufplan nicht enthalten und gehen verloren. Eine formale Kontrolle des geplanten Bauablaufs hinsichtlich seiner Vollständigkeit ist nur begrenzt möglich, da beispielsweise existierende Methoden der 4D-Visualisierung derzeit nicht ausreichend in den Prozess der Planung von Bauabläufen integriert sind.

Gegenstand der vorliegenden Arbeit ist die Entwicklung eines neuen Modells für die Unterstützung der Bauablaufplanung. Dafür wird der größtenteils manuelle Vorgang der Bauablaufplanung auf Basis verfügbarer Bauwerksinformationsmodelle (BIM) weitestgehend automatisiert und die Methodik der 4D-Animation in den Prozess der Bauablaufplanung integriert.
Ausgehend von in einer Erfahrungsdatenbank gespeicherten Informationen werden auf Basis einer Ähnlichkeitsermittlung Bauteilen des betrachteten BIM geeignete Vorgänge zugeordnet und mittels Algorithmen der Graphentheorie ein Workflowgraph aller möglichen Bauablaufvarianten generiert.
Aufgrund der vorgenommenen Kopplung des Bauablaufplans mit Bauteilen eines BIM und der visuellen Darstellung des Bauablaufs kann vom Planer im Rahmen der Modellierungsgenauigkeit des BIM auf die Vollständigkeit des Bauablaufplans geschlossen werden. Dies ermöglicht dem Anwender ein hohes Maß an Kontrolle des geplanten Bauablaufs bereits innerhalb der Planungsphase. Weiterhin unterstützt das entwickelte Modell die Integration von Ablaufvarianten, was deren Gegenüberstellung ermöglicht und die Wiederverwendbarkeit bereits geplanter Bauabläufe durch eine entsprechend ausgerichtete Abbildung des Modells.

Die Anwendbarkeit des erarbeiteten Modells wird anhand einer prototypischen Implementierung nachgewiesen und anhand eines Praxisbeispiels verifiziert.

Abstract

The efficient and targeted execution of construction works is greatly affected by the underlying construction schedule. By using common methods and models for construction scheduling, this process is usually error-prone and expensive.
In common practise, the focus of construction scheduling is limited to the documentation of the final result. Possible construction alternatives that were considered during the planning phase are not included in the resulting construction schedule and are lost for further consideration. A formal control of the planned construction schedule in terms of completeness is very limited, since existing methods such as 4D visualization are insufficiently integrated in the planning process.

The presented work addresses the development of a new model that supports the construction planning process. Therefore, the mostly manual process of construction planning will be largely automated based on available building information models (BIM) and integrates the methodology of 4D animation into this process.
Based on information stored in a knowledge database, appropriate tasks will be assigned to building elements of the considered BIM on the basis of similarity calculations. Subsequently, a workflow graph is generated by using algorithms of graph theory.
Due to the resulting logical interconnection of BIM objects and schedule processes and the visual representation of the construction building process, it becomes possible for the planner to suggest on the completeness of the construction schedule. This allows for a high degree of control over the planned construction schedule, already during the planning phase. Furthermore, the developed model supports the integration of construction alternatives into the schedule to allow their comparison as well as the reusability of already planned construction schedules by an appropriately targeted model mapping.

The developed model's applicability is shown based on a prototype implementation and verified using a practical example.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

„. . . Im „Leitbild Bau“ werden Durchgängigkeit der Wertschöpfungskette Bau, Partnerschaftliches Bauen, Betrachtung des Lebenszyklus, Bildung und Innovation als die Leitlinien des zukünftigen Bauens in Deutschland genannt. Eine wesentliche Umsetzungskomponente ist hierbei ein verbesserter Einsatz von IT. Durchgängigkeit der Wertschöpfungskette ist somit auch eine Durchgängigkeit der IT-Anwendung. Eine durchgängige IT-Nutzung auf der Basis von standardisierten Austauschverfahren und schrittweise auf Basis digitaler Bauwerksmodelle, die eine Lebenszyklusbetrachtung erst ermöglichen, muss daher angestrebt werden. . . . Die Umsetzung kann sich dabei nicht nur auf den IT-Aspekt konzentrieren, sondern umfasst auch eine innovative Weiterentwicklung der Arbeitsabläufe. Hierzu bedarf es der Bereitschaft der am Bau Beteiligten.“

Rüdiger Marquard, DIN Deutsches Institut für Normung e.V.
aus [Liebich u. Gersching 2010]

„. . . It’s the „I“ in BIM. When I talk to people really implementing BIM the same idea exists everywhere - that the reason to do BIM is to create a database of information that represents the design and [enables] digital organization. . . [and] there needs to be a very clear BIM value proposition for the owner.“

Jay Bhatt, Senior Vice President, AEC Solutions, Autodesk
aus [Jones 2010]

1.1 Problemstellung

Ausführungsprozesse im Bauwesen beinhalten viele Vorgänge und weisen komplexe Abhängigkeiten auf. Die Erstellung eines Bauablaufplans wird herkömmlich per Hand ausgeführt und dokumentiert lediglich das Endergebnis dieser Planung. Dabei erfolgt die Festlegung eines möglichen Bauablaufplans oft ohne Berücksichtigung anderer möglicher Leistungen. Gewöhnlich ist jedoch mehr als ein Bauablauf für das Erreichen eines Projektziels möglich und die Analyse von Alternativen wünschenswert.
Die aus der herkömmlichen Vorgehensweise resultierenden Bauabläufe sind festgeschrieben und unflexibel. Sich daraus ergebende Konflikte mit der Realität sind unvermeidbar. Die infolge dessen notwendigen Anpassungen der Bauablaufplanung werden jedoch

aufgrund des resultierenden Aufwands nicht getätigt, was mit einem hohen Verlust an Kontrolle des Bauablaufs einhergeht.
Erstellte Bauablaufplanungen sind aufgrund fehlender Modelle kaum wiederverwendbar und müssen in der Regel für jedes Projekt von Grund auf neu erstellt werden.
Moderne Möglichkeiten der 4D-Animation sind nicht in den Prozess der Bauablaufplanung integriert, sondern sind dem Prozess der Planung lediglich für deren Visualisierung nachgelagert. Eine solche virtuelle Darstellung des Bauablaufs bedarf ebenfalls einer aufwändigen Erstellung und händischen Nachführung des Modells als Reaktion auf Planungsänderungen.

1.2 Zielsetzung und Abgrenzung

Zielsetzung der Arbeit ist das Entwickeln eines Modells für die Unterstützung der Bauablaufplanung. Es wird die teilautomatisierte Generierung von Bauablaufplänen mit Ausführungsvarianten für die Berücksichtigung und Untersuchung dieser Varianten im Hinblick auf die Realisierung von Teilleistungen der Bauausführung angestrebt. Durch die Integration von Varianten in den Bauablauf werden zwei Ziele verfolgt. Zum einen soll die transparente Dokumentation von Entscheidungen ermöglicht werden, indem der Bauablaufplan mit den innerhalb der Planungsphase betrachteten Varianten diese reflektiert. Zum anderen wird eine schnelle Reaktion bei Störungen auf der Baustelle durch das Vorhalten von Ablaufvarianten erwartet.
Infolge der Teilautomatisierung des Planungsvorgangs wird eine schnelle und kontrollierte Anpassung des Bauablaufplans bei Modifikation der Bauwerksplanung durch erneutes Generieren des Bauablaufplans angestrebt.
Weiterhin wird das Ziel verfolgt, die Wiederverwendbarkeit bereits ausgeführter Bauablaufpläne zu ermöglichen und diese so als Wissensbasis für Planer verfügbar zu machen.
Für die angestrebten Ziele soll die Auswertung von in Bauwerksinformationsmodellen enthaltenen Informationen im Hinblick auf den Prozess der Bauablaufplanung untersucht werden. Unter diesem Gesichtspunkt wird die Integration einer 4D-Animation in den Prozess der Bauablaufplanung angestrebt. Durch die Integration einer 4D-Animation bereits in der Planungsphase des Bauablaufs wird eine verbesserte Kontrolle des zu planenden Bauablaufs gegenüber herkömmlichen Methoden erwartet.
Methoden für die Optimierung des Bauablaufs hinsichtlich des Einsatzes von Ressourcen, dem zeitlichen Ablauf oder anderen Gesichtspunkten der Netzplantechnik sind nicht Gegenstand dieser Arbeit. Vielmehr ist zu erwarten, dass diese Methoden eine sinnvolle Erweiterung für die Auswahl von im angestrebten Modell enthaltenen Ausführungsvarianten darstellen.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit ist in 7 aufeinander aufbauende Kapitel eingeteilt. Den Kern der Arbeit bilden die Kapitel 2.2 bis 2.5 sowie 4 und 5.

Kapitel 2 - Problem und Lösungsansatz
Nach der Einleitung wird in Kapitel 2 zunächst der derzeitige Stand der Forschung und Technik betrachtet. Es wird ein Überblick über den Stand gegenwärtiger Technologien, Methoden und Forschungsansätze bezogen auf die Bauablaufplanung gegeben. Aufbauend auf dieser Betrachtung wird die Problemstellung der Arbeit konkretisiert und ein entsprechender Lösungsansatz erarbeitet. Zu diesem Zweck werden die Defizite herkömmlicher Modelle der Bauablaufplanung identifiziert und diskutiert.

Kapitel 3 - Grundlagen und Begriffe
Im dritten Kapitel werden notwendige Grundlagen und Begriffe beschrieben, die für das Verständnis der nachfolgenden Kapitel wesentlich sind. Es handelt sich insbesondere um die Darstellung verschiedener Algorithmen für bipartite Graphen und dessen erweiterte Definitionsform den Workflowgraphen. Diese bilden das Fundament für die Entwicklung des angestrebten Modells im folgenden Kapitel.

Kapitel 4 - Generierung von Ablaufalternativen
Innerhalb dieses Kapitels wird das Modell für die teilautomatisierte Generierung von Bauablaufplänen entwickelt. Die Beschreibung des Modells erfolgt dabei allgemeingültig auf Basis der Mengenlehre und Relationenalgebra. Die zum Erreichen der Ziele entwickelten Algorithmen werden ebenfalls in allgemeiner Form dargestellt und detailliert beschrieben.

Kapitel 5 - Pilotimplementierung
Für das in Kapitel 4 entwickelte Modell wird dessen prinzipielle Anwendbarkeit in Form eines entwickelten Prototypen nachgewiesen. Wichtige Aspekte der Umsetzung werden dabei dargestellt und erläutert.

Kapitel 6 - Anwendungsbeispiel
Basierend auf der Pilotimplementierung des fünften Kapitels wird ein Anwendungsbeispiel vorgestellt. Es wird die prinzipielle Vorgehensweise für die Planung eines Bauablaufs auf Basis des im vorangestellten Kapitel beschriebenen Prototypen anhand eines Beispielprojekts gezeigt.

Kapitel 7 - Zusammenfassung und Ausblick
Die Arbeit schließt mit dem siebten Kapitel. Es werden die erzielten Ergebnisse in Kurzform zusammengefasst und kritisch diskutiert. Abschließend werden Anknüpfpunkte für eine mögliche Weiterführung des vorgestellten Forschungsansatzes gegeben.

2 Problem und Lösungsansatz

In diesem Kapitel wird zunächst der derzeitige Stand der Forschung und Technik bezogen auf die Bauablaufplanung betrachtet. Es vermittelt eine kompakte Übersicht über die verschiedenen Forschungsaktivitäten im Bereich der Bauablaufplanung der letzten Jahre und gibt Auskunft über derzeit in der Praxis eingesetzte Werkzeuge.
Aufbauend auf dieser Betrachtung wird die Problemstellung der Arbeit konkretisiert und ein entsprechender Lösungsansatz erarbeitet. Zu diesem Zweck werden die Defizite herkömmlicher Modelle der Bauablaufplanung identifiziert und diskutiert.
Es folgt die Betrachtung des Prozesses der Terminplanung. Aus den sich daraus ergebenden Schlussfolgerungen wird ein Lösungsansatz erarbeitet und vorgestellt.

2.1 Stand der Forschung und Technik

Eine aktuelle und sehr ausführliche Betrachtung der Forschungslandschaft, aber auch der Praxis zum Thema der Bauablaufplanung ist in [Tulke 2010] zu finden. Eine ebenfalls aktuelle Betrachtung mit dem Schwerpunkt der Wissensrepräsentation und Verarbeitung in der Bauablaufplanung ist in [Mikulakova 2010] dargestellt. In den genannten Arbeiten beschriebene als auch weiterführende Aspekte werden nachfolgend in zusammengefasster Form dargestellt. Für eine ausführliche Abhandlung, insbesondere zum Thema 4D-Animation und Terminplanung, wird ausdrücklich [Tulke 2010] empfohlen.

2.1.1 Bauwerksinformationsmodelle (BIM)

Obwohl jede Beschreibung eines Bauwerks (also auch die Beschreibung in Form von zweidimensionalen Plänen) im weitesten Sinne ein Bauwerksinformationsmodell darstellt, bezeichnet dieser Begriff hier die computergestützte Beschreibung eines Bauwerks in Form eines digitalen Datenmodells. Entgegen der reinen Dokumentation der Bauwerksplanung wird innerhalb solcher Modelle die Abbildung von Daten angestrebt, die den gesamten Lebenszyklus des Bauwerks betreffen. Entsprechende Anforderungen an ein solches Modell wurden in [Björk 1995] dargestellt. Eine Übersicht über die Entwicklung solcher Modelle beziehungsweise der Bauwerksmodellierung findet sich in [Eastman 1999].

Industry Foundation Classes (IFC)

Das bekannteste BIM stellen die IFC dar. Es handelt sich um ein objektorientiertes Modell zur Beschreibung von Bauwerken.

In der Praxis und in der Literatur wird die Rolle der IFC zum Teil kontrovers diskutiert. Es werden hierzu hauptsächlich zwei Meinungen vertreten. Einerseits werden die IFC lediglich als standardisiertes Datenformat für den Datei basierten Austausch betrachtet. Andererseits wird das objektorientierte Modell der IFC als Schema für die Implementierung eines zentrales Modells für alle am Bau Beteiligten beispielsweise auf Basis eines BIM-Servers herangezogen.
Der Beginn der Entwicklung des Modells geht auf das Jahr 1995 zurück. Es wird von der Organisation *buildingSMART*[1] definiert. Die Semantik des Objektmodells wird unter Verwendung der Sprache EXPRESS [ISO 2004] definiert und ist frei verfügbar[2]. Die derzeitige verfügbare Version wird mit IFC2x3 TC1 bezeichnet und wurde im Juli 2007 veröffentlicht. Die Veröffentlichung der nachfolgenden Version IFC2x4 ist für Anfang 2011 geplant [buildingSMART 2008]. Das IFC Format ist bei ISO[3] als ISO/PAS 16739[4] registriert und ist in Begriff unter ISO/IS 16739 ein internationaler Standard zu werden [Liebich u. Gersching 2010].

Verfügbare Software Export und Import von Bauwerksmodellen werden inzwischen von einer Vielzahl an Software-Applikationen unterstützt. Tabelle 2.1 zeigt eine Übersicht der offiziell nach IFC2x3 Stufe 2 zertifizierten Anwendungen, die sowohl den Import als auch den Export von IFC-Daten unterstützen.

Produkt	Firma	Webseite
ALLPLAN 2006.2	Nemetschek	www.nemetschek.de
ArchiCAD 11	Graphisoft	www.graphisoft.de
AutoCAD Architecture 2008 SP1	Autodesk	www.autodesk.de
Bentley Architecture 8.9.3	Bentley Systems	www.bentley.com
DDS-CAD 6.4	DDS	www.dds-cad.de
EliteCAD Architecture 11 SP1	Messerli Informatik	www.elitecad.eu
Revit Architecture 2008 SP1	Autodesk	www.autodesk.de
SCIA ESA-PT	Nemetschek	www.scia-software.de
TEKLA Structures 13.0	TEKLA Corporation	www.tekla.com

Tabelle 2.1: Software mit IFC-Import / Export Funktionalität - Quelle: http://www.buildingsmart.de/2/2_01_03.htm#IFC2x3Step2 - Abruf: 05.02.2011

Stufe 2 entspricht nach buildingSMART einer endgültigen Zertifizierung der Software. Sie wird erteilt, sobald die Verarbeitung komplexer IFC-Dateien nachgewiesen werden kann.

[1] ehemals Industrieallianz für Interoperabilität e.V. - steht für eine not-for-profit Organisation mit dem Ziel „... den modellbasierten Ansatz für die Optimierung der Planungs-, Ausführungs-, und Bewirtschaftungsprozesse im Bauwesen im Rahmen der buildingSMART-Initiative (Stichwort: Bauwerksmodell) zu etablieren.“, Quelle: http://www.buildingsmart.de/1/index.htm - Abruf: 03.02.2011

[2] http://buildingsmart-tech.org/products/ifc_specification/ifc-releases/ifc2x3-tc1-release/summary - Abruf: 03.02.2011

[3] Internationale Organisation für Normung

[4] PAS - Publicly Available Specification, (engl.: öffentlich verfügbare Spezifikation)

Datenformate Der Austausch beziehungsweise die Bereitstellung von IFC-Daten kann auf verschiedene Arten erfolgen. Am weitesten verbreitet sind STEP-Dateien [ISO 2002a]. Weitere Möglichkeiten sind der Austausch als XML-Dateien [ISO 2002b] sowie der Austausch über IFC-Modellserver. Die zuletzt genannte Möglichkeit findet häufig Anwendung in der Forschung im Hinblick auf den kollaborativen Einsatz von BIM (vgl. [Plume u. Mitchell 2007] und [Chen u. a. 2005]). Als kommerzielle Lösungen existieren der *Eurostep Model Server* (Eurostep - www.eurostep.com) und der *EDModelServer for IFC* (EPM Technology AS - www.epmtech.jotne.com). Eine verfügbare open source Umsetzung eines BIM-Servers auf Basis der IFC ist unter http://bimserver.org/ zu finden (vgl. [Beetz u. a. 2010]).

Bedeutung in der Praxis Bauwerksinformationsmodelle gewinnen in der Praxis zunehmend an Bedeutung, so ist ein Übergang von der Nutzung traditioneller CAD-Software zu BIM-Software erkennbar (vgl. [Watson 2010] und [Howard u. Björk 2008]). Insbesondere die IFC spielen hierbei eine bedeutende Rolle. Beispielsweise haben die Verwaltungsbehörden für öffentliche Gebäude der USA GSA[5], Dänemark, Finnland und Norwegen eine Absichtserklärung für den Einsatz und die Unterstützung der Weiterentwicklung offener BIM Standards, konkret der IFC, unterzeichnet (siehe [GSA 2008]). Auch in Deutschland gewinnt der Einsatz der IFC an Bedeutung. Beispielsweise setzt die Staatliche Bauverwaltung Bayerns verstärkt die IFC innerhalb ihrer Datenverwaltung ein (siehe [Pries 2007]). Es ist zu erwarten, dass sich dieser Trend nach erfolgter Standardisierung der IFC als ISO Norm (siehe [Liebich u. Gersching 2010]) weiter verstärkt. Das Interesse an einem offenen BIM Standard begründet sich an dieser Stelle in der Erwartung eines höheren Nutzwertes der erstellten Bauwerksinformationen über den gesamten Lebenszyklus eines Bauwerks. Die Ablage dieser Informationen als Sammlung von Dokumenten wie Zeichnungen, Listen usw. wird den heutigen Anforderungen zunehmend weniger gerecht, da deren Auswertung nicht formal möglich ist. Gleichzeitig sichert ein offenes BIM-Format die zukünftige Lesbarkeit der darin abgebildeten Informationen.

Weitere Modelle

Im direkten Vergleich mit den IFC als offenes BIM existiert keine konkrete Alternative. Zwar beinhaltet jede Applikation, die eine Planung von Bauwerken auf Modellbasis erlaubt (siehe Tabelle 2.4), ein Bauwerksinformationsmodell, der direkte Zugriff auf die enthaltenen Informationen ist jedoch nur über die entsprechende Applikation oder gegebenenfalls über Schnittstellen möglich. Diese Modelle sind jeweils auf die zugrunde liegende Applikation zugeschnitten und somit nicht standardisiert, was die inhaltliche Auswertung der darin abgebildeten Informationen erschwert.
In der Forschung existierende generische Modelle bieten prinzipiell die Möglichkeit der Abbildung eines BIM. Dahingehende Forschungsansätze stellen beispielsweise die Möglichkeit der Zusammenfassung und Verwaltung von Teilmodellen verschiedener Disziplinen der Bauwerksplanung und des Bauwerksmanagements zu einem Zentralmodell

[5]GSA - U.S. General Services Administration

bereit. Bedingt durch den generischen Ansatz müssen derartige Modelle zunächst entsprechend den Anforderungen des Abbildungszwecks initialisiert beziehungsweise implementiert werden, was ein entsprechendes Expertenwissen voraussetzt (vgl. [Wender 2009], [Hauschild 2003] und [Willenbacher 2002]). Auch hier ist die Möglichkeit der Auswertung im Modell abgebildeter Informationen von der Implementierung beziehungsweise Initialisierung des generischen Modells abhängig.

2.1.2 Terminplanung

In [Winch u. Kelsey 2005] wird im Jahr 2005 unter anderem die Frage aufgegriffen, welche Methoden Terminplaner in der Praxis nutzen. Für die Beantwortung dieser Frage wurden achtzehn Planer von fünf nicht näher bezeichneten, in England führenden Firmen nach ihren täglichen Aufgaben befragt. Als etablierte Methode wurde der Einsatz von Netzplantechnik [DIN 2009] unter Verwendung der *Critical Path Method* (CPM) oder *Program Evaluation and Review Technique* (PERT) auf Basis von Balkendiagrammen als Standardverfahren identifiziert. Seinerzeit eingesetzte Software-Applikationen waren in der Reihenfolge ihrer Wichtigkeit Primavera, Power Project und in Ergänzung MS Excel. In [Besner u. Hobbs 2008] aus dem Jahr 2008 wird unter anderem erneut ein Querschnitt der eingesetzten Methoden der Terminplanung ermittelt. Auch hier wird die Netzplantechnik sowie die CPM als wichtigstes Verfahren der Terminplanung genannt und entspricht dem derzeitigen Standardverfahren.

Die Hauptziele der Netzplantechnik beinhalten die Ermittlung der Dauer eines Projekts, die Identifikation möglicher Risiken und die Bestimmung kritischer Vorgänge im Bauablauf, welche die Gesamtdauer des Bauablaufs maßgeblich beeinflussen. Mit diesem Wissen können beispielsweise nicht kritische Vorgänge umgeplant werden, ohne den Gesamtablauf zu stören.

Die CPM ist hierbei als deterministisches Verfahren anzusehen. Beispielsweise verlangt die Bewertung von Vorgängen mit Dauern die exakte Angabe eines frühesten Start- und Endtermins. Dieser Umstand ist problematisch, da die Angabe derartiger Informationen naturgemäß mit Ungewissheit behaftet ist.

Die PERT ist als nichtdeterministisches Verfahren einzustufen, da an dieser Stelle mit der Angabe von wahrscheinlichen Dauern gearbeitet wird. Es existieren weitere Ansätze, die Ungewissheit über die Angabe von Dauern in die Berechnung einfließen zu lassen. Beispielsweise wird in [Freundt 2005; Freundt u. Beucke 2004] die Angabe von Dauern auf Basis der Fuzzy-Logic[6] untersucht. Ziel ist eine gegenüber deterministischen Modellen robustere Modellierung des Bauablaufs, um die Auswirkungen von Störungen in der Bauausführung, auf den geplanten Bauablauf zu minimieren.

Die Netzplantechnik und deren Methoden werden in der Literatur weitläufig behandelt. Weiterführende Informationen finden sich beispielsweise in [Kochendörfer u. a. 2010], [Zimmermann u. a. 2010] und [Domschke u. Drexl 2007].

[6] „verschwommene Logik“ - Modellierung von Unschärfen auf Basis umgangssprachlicher Beschreibungen - vgl. [Böhme 1993]

Software für die Terminplanung In [Tulke 2010] wurden die vier in Tabelle 2.2 dargestellten Software-Applikationen, als die den Markt anführenden Terminplanungsprogramme, basierend auf Interviews identifiziert. Alle Applikationen bieten die Planung nach der CPM mit dem Ergebnis eines Balkenplans. Die Applikation *ViCo Control* bietet weiterhin die Planung auf Basis der *Line of Balance*[7] (LoB) Methode.

Firma	Internetseite	Produktname	Verfahren
Asta	www.teamplan.co.uk	Teamplan	CPM
Microsoft	officemicrosoft.com	MS Project	CPM
Oracle	www.primavera.com	Primavera P6	CPM
VICO Software	www.vicosoftware.com	ViCo Control	LoB

Tabelle 2.2: Überblick über die meist verbreiteten Terminplanungsprogramme - Quelle: [Tulke 2010]

2.1.3 4D-Animation

Eine weiteres Gebiet der Terminplanung betrifft die 4D-Animation, bei welcher der geplante Bauablauf visualisiert wird.

Bedeutung in der Forschung In der Forschung ist die virtuelle Darstellung des Bauablaufs das in den letzten Jahren am weitläufigsten bearbeitete Gebiet in der Bauablaufplanung. Dieser Umstand ist insbesondere auf die zunehmende Verfügbarkeit dreidimensionaler Bauwerksplanungen zurückzuführen, welche die Bauwerksgeometrie darstellen. Die vierte Dimension betrifft die zeitliche Bewertung von Vorgängen des Bauablaufs, deren Abfolge auf dieser Basis virtuell dargestellt wird. Grundlage dafür ist die Verknüpfung des dreidimensionalen Gebäudemodells mit einem fertig geplanten Bauablauf. In diesem Zusammenhang wird sowohl in der Literatur als auch in der Praxis von einer 4D-Animation bzw. 4D-Simulation gesprochen.

Es ist anzumerken, dass oft zwischen beiden Begriffen nicht unterschieden wird. Der Begriff Simulation wird dabei häufig auch für eine reine Animation des Bauablaufs verwendet. Gleichwohl gibt es weiterführende Ansätze, die dem Begriff der Simulation eher entsprechen. Beispiele dafür sind die Simulation von im Bauablauf eingesetzten beweglichen Kränen für die Erkennung von örtlichen Kollisionskonflikten beziehungsweise gegenseitigen Behinderungen, die den Bauablauf stören [Tantisevi u. Akinci 2009] oder die Optimierung eines Bauablaufs hinsichtlich verschiedener Kriterien durch dessen Constraint-Basierte Simulation [König u. Beißert 2009]. Aus diesem Grund wird innerhalb dieser Arbeit der Begriff 4D-Animation verwendet.

Zweck einer solchen Darstellung ist sowohl die visuelle Kontrolle als auch die Kommunikation des geplanten Bauablaufs. Der Nutzen ist die bessere Verständlichkeit des

[7] Weg-Zeit-Diagramm, bei der die Fertigstellung von Bauabschnitten bezogen auf die Zeit dargestellt wird - vgl. [Uher 2003].

Bauablaufs gegenüber der sonst üblichen Kommunikationsform als Balkenplan und in diesem Zusammenhang die Vermeidung von logischen Fehlern im Bauablauf. Ein weiterer Vorteil besteht in der daraus resultierenden räumlichen Einordnung von Aufgaben des Bauablaufplans in das Bauwerksmodell. Untersuchungen zur Anwendbarkeit und dem Nutzen von 4D-Planungen sind in [Koo u. Fischer 1998] dargelegt.
Bauteile des geplanten Bauwerks werden zu diesem Zweck entsprechend ihren zugeordneten Vorgängen ein- oder ausgeblendet. Unterstützt wird diese Darstellung durch eine entsprechende Einfärbung der Bauteile, um deren Status (im Bau befindlich, fertig, usw.) zu visualisieren. Diesbezügliche Untersuchungen sind im Hinblick auf die Art der Darstellung in [Chang 2009] und [Koo u. Fischer 2003] beschrieben. Ergebnis der Untersuchungen ist unter anderem die Empfehlung, eine einheitliche Darstellungsform für eine 4D-Animation zu definieren, um deren Transparenz für den Anwender zu erhöhen.
Die herkömmliche Vorgehensweise für die Erstellung einer 4D-Animation besteht in der händischen Verknüpfung von Vorgängen des Bauablaufplans mit Objekten des Bauwerksmodells.
An dieser Stelle ist bereits ein grundsätzliches Problem dieser Vorgehensweise zu benennen. Das aus der CAD-Planung stammende Bauwerksmodell weist in der Regel nicht die notwendige Granularität bzw. Strukturierung auf, die für eine korrekte Verknüpfung notwendig ist. Beispielsweise wird eine Betondecke durch aneinandergereihte Betonierabschnitte hergestellt, diese im CAD-System jedoch meist durchgängig modelliert. Diesem Umstand kann zwar mit einer groben Modellierung des Bauablaufplans begegnet werden, geht aber mit einer daraus resultierenden groben Animation einher. Alternativ ist das Modell im CAD-System entsprechend den Vorgaben der Bauablaufplanung anzupassen, was den Aufwand für die Erstellung einer 4D-Animation erhöht und die enge Zusammenarbeit zwischen Konstrukteur und Terminplaner erforderlich macht. Die zweite Methode führt zwangsläufig zu Problemen, falls mehrere Bauabläufe für dasselbe Gebäudemodell erstellt werden müssen, was aus verschiedenen Gründen notwendig werden kann (siehe [Robert 2008] und [Koo u. Fischer 2003]).
Die manuelle Methode der Verknüpfung ist gleichzeitig mit einem hohen Aufwand verbunden, wie auch in [Koo u. Fischer 1998] und [Koo u. Fischer 2000] wiedergegeben, da Vorgänge und dazu korrespondierende Bauwerksobjekte manuell selektiert und verknüpft werden müssen. Forschungsthemen im Bereich der 4D-Animation beschäftigen sich aus diesem Grund vorwiegend mit der Optimierung des Vorgangs für die Erstellung einer 4D-Animation. Das konkrete Problem besteht dabei darin, den Aufwand für die notwendige Verknüpfung der Vorgänge des Bauablaufs mit den Bauteilen des 3D-Gebäudemodells zu minimieren. [Tulke 2010] nennt neben der händischen Verknüpfung vier weitere Ansätze für die Erstellung von 4D-Animationen. Die in der Arbeit diesbezüglich enthaltenen Erkenntnisse werden nachfolgend zusammengefasst:

> *„Zuweisung von Terminen oder Bestimmung einer Erstellungsreihenfolge von Bauteilen in der CAD-Anwendung und anschließend automatische Erzeugung des Terminplans aus dem CAD-Programm.„*
>
> aus [Tulke 2010]

Als problematisch wird bei Verwendung dieser Methode festgestellt, dass die Definition von Vorgängen innerhalb der CAD-Anwendung stattfindet. An dieser Stelle findet eine Überschneidung der Aufgabenbereiche des Konstrukteurs und des Terminplaners statt. Vorgänge des Bauablaufs müssen demnach bereits innerhalb der Konstruktionsplanung definiert werden oder aber erfordern eine enge Kooperation zwischen Konstrukteur und Terminplaner. Weiterhin ist die Vorgabe einer Reihenfolge von Vorgängen nicht effizient möglich. Eine entsprechende Umsetzung mit der Möglichkeit des Exports nach MS Project existiert beispielsweise als Entwicklung von Autodesk Consulting, basierend auf Autodesk Revit [Autodesk-Consulting 2007].

> *„Erzeugung einer gleichartigen Baumstruktur des Bauwerksmodells (product break down structure) und des Terminplans (work break down structure). Anschließend erfolgt eine Abbildung der beiden Bäume aufeinander.„*
>
> aus [Tulke 2010]

Hierbei handelt es sich um Forschungsansätze, die in [Aalami u. a. 1998], [Dawood u. a. 2003], [Kang u. a. 2005] und [Dawood u. a. 2005] dargestellt sind. Es wird eine Automatisierung erreicht, indem das Bauwerksmodell und der Terminplan einer Gliederungsstruktur unterworfen wird, was eine anschließende Abbildung beider Modelle aufeinander erlaubt. An dieser Stelle werden die dadurch entstehenden Zwänge aufgrund der gegebenen Gliederungsstruktur als zu unflexibel eingeschätzt.

> *„Nutzung von Datenbankanfragen zur Selektion von Bauteilen auf Basis des Objekttyps und vergebener Attribute.“*
>
> aus [Tulke 2010]

Bei dieser Methode wird der Aufwand für die Selektion von typisierten Objekten des Bauwerksmodells verringert beziehungsweise kann gezielt auf Basis von Objekteigenschaften erfolgen. Die Eigenschaften eines Objekts sind dabei abhängig vom zu Grunde liegenden Bauwerksmodell. Anschließend erfolgt die Zuweisung des Ergebnisses einer Datenbankanfrage (Menge von Objekten des Bauwerksmodells) zu Vorgängen des Bauablaufplans. Vorgang und Objekte des Bauwerks werden dabei automatisch verknüpft. Hervorzuheben ist, dass durch die Verwendung von Abfragen anhand von Objekteigenschaften der Zuordnungsprozess vom konkreten Bauwerksmodell entkoppelt wird. Abfragen können entsprechend auch für andere Bauwerksmodelle genutzt werden. Weiterhin können Änderungen im Bauwerksmodell auf diese Weise in gewissem Umfang berücksichtigt werden. Die genannte Methode ist in der Applikation Autodesk Navis-Works Jetstream umgesetzt. An dieser Stelle wird das Fehlen einer Abfragemöglichkeit, bezogen auf die räumlichen Lage von Objekten des Bauwerksmodells, als problematisch festgestellt.

> *„Interaktive Erstellung des Terminplans in einer 3D-Umgebung mit gleichzeitiger Spezifikation der Visualisierung für 4D.“*
>
> aus [Tulke 2010]

Der letzte Ansatz entspricht einer interaktiven Erstellung der 4D-Animation infolge des virtuellen Zusammensetzens des Bauwerksmodells durch den Anwender. In [Zhou u. a. 2009] und [Waly u. Thabet 2002] werden Bauteile ähnlich dem Baukastenprinzip Schritt für Schritt, entsprechend dem bestehenden Bauwerksmodell, erneut zusammengefügt. Die zugehörigen Vorgänge werden bei jedem dieser Schritte definiert, bewertet und dem jeweiligen Bauteil zugeordnet. Auf diese Weise entsteht mit der Definition des Bauablaufs die zugehörige 4D-Animation, die zu jedem Zeitpunkt visualisiert werden kann. Durch diesen Ansatz wird die 4D-Animation in den Prozess der Bauablaufplanung integriert. Im Grunde entspricht dies der manuellen Zuordnung eines Vorgangs zu jedem Bauteil. Die sequentielle Reihenfolge von Vorgängen ergibt sich jedoch dabei automatisch anhand der durch den Anwender vorgegeben Reihenfolge des Zusammensetzens. An dieser Stelle ergibt sich im Vergleich zur händischen Verknüpfung ein ähnlich hoher Aufwand für die Erstellung des Terminplans respektive der 4D-Animation.

In [Tulke 2010] selbst wird verschiedenen, hier angesprochenen, Problemen Rechnung getragen. Basierend auf der Nutzung von Datenbankanfragen wird für die Selektion von Bauteilen auf Basis von Objekteigenschaften eine umfassende Abfragesprache entwickelt, welche die genannte Methode um Abfragemöglichkeiten hinsichtlich der räumlichen Lage von Objekten auf Basis geometrischer Methoden erweitert. Es wird dabei die Lage von Objekten des Bauwerksmodells bezogen auf deren Lage zu geometrisch definierten Arbeitszonen und Achsen berücksichtigt. Gleichzeitig wird die geometrische Teilung von Objekten des Bauwerksmodells anhand von Arbeitszonen und Achsen unterstützt, was die daraus resultierende 4D-Animation von der innerhalb der CAD-Applikation konstruierten Geometrie des Bauwerksmodells entkoppelt. Dadurch wird ein wesentliches Problem beim Erstellen von 4D-Animationen gelöst. Bisher notwendige Absprachen zwischen Terminplaner und CAD-Konstrukteur entfallen, die bisher für das Erreichen der für die 4D-Animation notwendigen Objektgranularität notwendig waren.
Tabelle 2.3 fasst die in der Literatur benannten Prototypen mit 4D-Funktionalität zusammen:

Organisation	Anwendungsname	Literaturquelle
University of Salford	web-based 4D/IFC data base	[Tanyer u. Aouad 2005]
University of Teeside	4D/VR information base	[Dawood u. a. 2003], [Dawood u. a. 2005]
VTT	Visual Product Chronology	[Kähkönen u. Leinonen 2003], [Porkka u. Kähkönen 2007]
University of Wolverhampton	4DX	[Zhou u. a. 2009]
Carnegie Mellon University	SiDaCoS	[Jan Reinhardt 2004]
University of British Columbia	REPCON	[Staub-French u. a. 2008]
Bauhaus-Universität Weimar, HOCHTIEF	Scheduling Assistant	[Tulke 2010]

Tabelle 2.3: 4D-Animationssoftware in der Literatur - nach [Tulke 2010]

Software für die 4D-Animation Tabelle 2.4 gibt einen Überblick über kommerziell verfügbare 4D-Animationsprogramme. Laut [Tulke 2010] handelt es sich bei Autodesk Navisworks und Synchro Project Constructor um die derzeit am Markt etabliertesten Produkte. Ein Vergleich der genannten Applikationen hinsichtlich des Funktionsumfangs wird jedoch nicht dargestellt. Hierzu wird auf [Heesom u. Mahdjoubi 2004], [Leen-Seok u. a. 2004], [Sheppard 2004], [Porkka u. Kähkönen 2007] und [Eastman u. a. 2008] verwiesen.

Firma	Internetseite	Produktname
Autodesk	www.autodesk.com	Revit Architecture, Navisworks Simulate
Balfour Technologies	www.bal4.com	fourDscape
Bentley	www.bentley.com,	ProjectWise Navigator
Bentley	www.commonpointinc.com	ConstructSim
Building Explorer	www.buildingexplorer.com	Building Explorer
Ceco Interactive Design	www.ceco.se	Ceco Viewer
D-studio	www.d-studio.be	xD Virtual Builder
iD-ProPlan	www.idproplan.co.uk	4D-CCIR
Gehry Technologies	www.gehrytechnologies.com	Digital Project
Innovaya	www.innovaya.com	Visual Simulation
Intergraph	www.intergraph.com	SmartPlant Review
Synchro	www.synchroltd.com	Project Constructor
Tekla	www.tekla.com	Tekla Structures
VICO Software	www.vicosoftware.com	ViCo 5D Presenter

Tabelle 2.4: Marktüberblick 4D-Animationssoftware - Quelle: [Tulke 2010]

2.1.4 Generierung von Bauablaufplänen

Erste Forschungsansätze für die Generierung von Bauabläufen sind in zeitlicher Abfolge ihrer Veröffentlichung *CONSTRUCTION PLANEX* [Hendrickson u. a. 1987], *ORA-PLAN* [Darwiche u. a. 1988], *GHOST* [Navinchandra u. a. 1988], *Builder* [Cherneff u. a. 1991], *KNOW-PLAN* [Morad u. Beliveau 1994] und *CasePlan* [Dzeng u. Tommelein 1997; Dzeng 2000; Dzeng u. Whang 2003; Dzeng u. Tommelein 2004].

Eine Auswahl der genannten Forschungsansätze wird ab Beginn der neunziger Jahre nachfolgend kurz dargestellt:

Builder Der in [Cherneff u. a. 1991] präsentierte Prototyp *Builder* erlaubt die Generierung von Bauabläufen auf Basis von Bauteilen und definierten Regeln. Hierzu wird ein eigens entwickeltes, wissensbasiertes 2D-CAD-System namens *Draw KM*[8] implementiert. *Draw KM* greift für das Erzeugen von Konstruktionsplänen auf eine Objekt-Datenbasis zurück. Diese beinhaltet verschiedene Klassen von Bauelementen

[8]Draw KM - draw knowlede module: wissensbasiertes Zeichenmodul

wie Wände und Türen. Methoden für die Darstellung der Objekte, die Netzplangenerierung, die Erzeugung von Balkendiagrammen und der Kostenkalkulation werden innerhalb dieser Klassen bezogen auf das zu repräsentierende Bauteil implementiert. *Draw KM* erzeugt ein semantisches Netz, welches die Abhängigkeiten von Objekten zueinander darstellt, beispielsweise die Zugehörigkeit einer Tür zu einer Wand. Ein weiteres Modul *Planner KM*[9] erzeugt auf Basis des semantischen Netzes und vordefinierter Regeln schließlich einen Netzplan.

KNOW-PLAN In [Morad u. Beliveau 1994] wird die Implementierung eines komplexen Geometrie basierten Planungssystems für Bauabläufe mit der Bezeichnung *KNOW-PLAN* vorgestellt. Zusammenfassend kann die Funktionsweise von *KNOW-PLAN* wie folgt dargestellt werden: Das System nutzt die geometrische Lage von Bauwerksobjekten im dreidimensionalen Raum, deren Abhängigkeiten zueinander sowie die räumliche Beziehung zu anderen Bauwerksobjekten als Basis für die Generierung eines zugehörigen Bauablaufs. Das 3D-Modell wird dazu über entsprechende Interfaces aus CAD-Daten (IGDS, 3DM)[10] in ein internes Format (WALKTHRU-Modell) überführt. Dabei werden die geometrisch räumlichen Beziehungen zwischen Bauwerksobjekten aus dem geplanten 3D-Modell des Bauwerks extrahiert. Die Zuordnung von Vorgängen zu Bauwerksobjekten erfolgt auf Basis von Objektklassen. Die Definition einer solchen Objektklasse wird vom Anwender vorgenommen und auf Basis einer festgelegten Menge möglicher Attribute beschrieben - welche Attribute an dieser Stelle möglich sind, wird jedoch nicht dargestellt. Als typische Attribute einer Klasse werden lediglich ein Name, mögliche Arbeitsrichtungen (x-, y- oder z-Richtung) und die Priorität der Arbeitsrichtung genannt, die darstellen soll, welche der drei möglichen Arbeitsrichtungen die bevorzugte ist. Weiterhin müssen vom Anwender Abhängigkeiten im Sinne ihrer Reihenfolge im Bauablauf zwischen den Objektklassen selbst angegeben werden. Ferner wird die Definition von geometrisch definierten Zonen und somit darin enthaltener Bauwerksobjekte unterstützt, deren Reihenfolge durch den Anwender vorgegeben werden kann. Der Bauablauf wird anhand von mehrstufigen, vordefinierten Regeln auf Basis geometrischer Informationen, der Arbeitsrichtung und vom Anwender definierter weiterführender Regeln generiert.

CasePlan Im Gegensatz zu den bereits genannten Ansätzen liegt der Fokus in [Dzeng u. Tommelein 1997] weniger auf dem Aspekt der Generierung von Bauabläufen als auf deren Wiederverwendung. Basierend auf der Methodik des Case Based Reasoning[11] (CBR) wird ein System vorgestellt, das die automatisierte Bauablaufplanung im Bereich des Anlagenbaus zur Errichtung von Dampfkesseln für Kraftwerke erlaubt. CBR ist ein Teilgebiet der Künstlichen Intelligenz (KI) und stellt ein maschinelles Verfahren zur Problemlösung durch Analogieschluss dar[12] (siehe [Aamodt u. Plaza 1994]).

[9] Planner KM - planner knoewledge modul: wissensbasiertes Planungsmodul

[10] IGDS - Intergraph Datei; 3DM - Bechtel Intergraph Datei

[11] engl.: Fall basiertes Schließen

[12] Wikipedia, http://de.wikipedia.org/wiki/Case-Based_Reasoning - Abruf: 01.02.2011), weiterführende Informationen zum Verfahren des CBR finden sich beispielsweise in [Kolodner 1993]

Grundgedanke dieses Verfahrens ist die Übertragbarkeit vorhandener Lösungen eines Problems auf ähnliche Probleme. Zu diesem Zweck wird in *CasePlan* der zu errichtende Dampfkessel als Problem und ein geplanter Bauablauf als dessen Lösung betrachtet. Diese Paare werden innerhalb einer Wissensbasis gespeichert. Infolge dieser Abbildung ist es unter Verwendung des Systems möglich Bauabläufe (als Lösung) auf Basis des CBR für ähnliche Dampfkessel (Problem) zu bestimmen und anzupassen. Ergebnis des Vorgangs ist ein Vorgangsnetzplan für die Errichtung des Dampfkessels. In späteren Arbeiten wird dieser Ansatz weiter verfolgt. In [Dzeng u. Whang 2003] wird aufbauend auf [Dzeng 2000] die Anwendung des *CasePlan* Systems anhand von Straßenbauprojekten gezeigt. In [Dzeng u. Whang 2003] erfolgt schließlich die Adaption des Ansatzes auf generische Produktmodelle. Die für die Anwendung des CBR notwendige Berechnung der Ähnlichkeit von Problemstellungen erfolgt hierbei auf Basis von Objekteigenschaften und wird in [Dzeng u. Tommelein 2004] ausführlich behandelt.

In [Mikulakova 2010] wird ebenfalls ein Konzept auf Basis des CBR für die Wiederverwendung und als Grundlage für die Generierung von Bauablaufplänen vorgestellt, das auf der automatisierten Zuordnung von Vorgängen gespeicherter Bauabläufe zu Bauteilen eines BIM beruht. Insbesondere der darin präsentierte Ansatz für die Ähnlichkeitsbestimmung von Bauteilen eines BIM findet im weiteren Verlauf der Arbeit Verwendung und wird an entsprechender Stelle erläutert.

Weitere Generierungsansätze An der TU-Berlin wurde ebenfalls ein Forschungsansatz vorgestellt, der dem Bereich der Generierung von Bauabläufen zuzuordnen ist. Hierzu existiert eine Reihe von Veröffentlichungen [Huhnt 2005], [Huhnt u. Enge 2006], [Enge u. Huhnt 2008], [Huhnt 2009] und [Enge 2010]. Der Grundgedanke des vorgestellten Ansatzes basiert dabei im Wesentlichen auf dem in [Heinrich u. Huhnt 2003] vorgestellten Prinzip der Modellierung von Vorgängen des Bauablaufs mit Voraussetzungen und Resultaten. Eine solches Konstrukt wird als *Template*[13] bezeichnet. Ein *Template* beschreibt hierbei alle notwendigen Vorgänge, die zum Erstellen, dem Umbau oder Abriss einer bestimmten Bauwerkskomponente notwendig sind. Nach der Zerlegung eines Bauwerks (die Granularität der Zerlegung ist dabei vom Nutzer zu wählen) in seine Komponenten wird diesen ein entsprechendes *Template* durch den Anwender zugeordnet. Anschließend werden weitere technologische Abhängigkeiten zwischen Vorgängen (resultierend aus den zugewiesenen *Templates*) verschiedener Komponenten definiert. Diese werden in Form von Vorgangsvoraussetzungen und Vorgangsresultaten modelliert. Die Abhängigkeiten sind vom Anwender vorzugeben. Voraussetzungen und Resultate beschreiben dabei Bauwerkskomponenten, die sich in einem bestimmten Zustand befinden. Auf Basis dieser Informationen wird ein Bauablauf generiert, indem die Voraussetzungen dieser Vorgänge mit Vorgängen passender Bauwerkskomponenten verknüpft werden. Der resultierende Bauablauf kann nach einer durch den Anwender erfolgten Bewertung der Vorgänge für eine 4D-Animation genutzt werden (siehe [Huhnt 2010]). Der Fokus des Ansatzes liegt hierbei weniger auf der Generierung des Bauablaufs, sondern mehr auf dessen Vollständigkeit und logischen Richtigkeit.

[13] engl.: Vorlage

Ein weiterer Ansatz wird in [de Vries u. Hanrik 2007] vorgestellt. Hierbei steht die Sequenzierung von Vorgängen des Bauablaufs im Blickpunkt. Um die Reihenfolge der Erstellung von Bauwerkskomponenten zu bestimmen, wird das einer CAD-Anwendung entstammende dreidimensionale Bauwerksmodell geometrisch analysiert. Dabei wird die relative Position von Bauwerkskomponenten untereinander untersucht. Der Ansatz setzt hierbei voraus, dass Komponenten, die sich geometrisch gesehen unterhalb von anderen Komponenten befinden, vor diesen erstellt werden müssen. Weiterhin werden zusätzliche Positionen bei der Analyse in Betracht gezogen (beispielsweise links daneben, rechts daneben, dahinter oder davor), die einer Priorisierung entsprechend dem Ansatz von [Morad u. Beliveau 1994] unterliegen. Auf diese Weise wird auf eine Reihenfolge der Erstellung von Bauwerkskomponenten geschlossen. Das Ergebnis dieser Sequenzierung kann anschließend in ein Terminplanungsprogramm importiert und weiterverarbeitet werden.

2.1.5 Zusammenfassung

4D-Animation Die 4D-Animation wird bereits praktisch angewendet und ist in der Forschung die im Moment am häufigsten untersuchte Methodik im Bereich der Bauablaufplanung. Dabei akzeptieren bestehende Forschungsansätze die in der Praxis übliche Arbeitsweise und untersuchen die Optimierung des Verknüpfungsvorgangs zwischen Objekten des dreidimensionalen Bauwerksmodells mit den Vorgängen eines erstellten Terminplans. Die Nachführung des Bauablaufplans bei Änderungen im Bauwerksmodell ist ebenfalls Gegenstand der Forschung. Eine Integration der 4D-Animation in den Prozess der Planung des Bauablaufs findet dabei nicht statt.

Generierung Methoden zur automatisierten Bauablaufplanung werden in der Praxis aufgrund fehlender Software nicht angewendet. Forschungsansätze zur Generierung von Bauabläufen existieren zum Teil als prototypische Umsetzungen. Bestehende Ansätze integrieren dafür die Zuordnung von Vorgängen zu Bauteilen in die CAD-Applikation [Cherneff u. a. 1991]. Den in der CAD-Applikation zur Konstruktion verfügbaren Bauteiltypen wird dafür im Vorfeld ein entsprechender Vorgang vom Anwender zugeordnet. Durch die Verwendung dieser Bauteiltypen in der Konstruktionsphase kann auf die notwendigen Vorgänge im Bauablauf geschlossen werden. Die Sequenzierung der Vorgänge erfolgt dabei in der Regel manuell. Bei diesem Ansatz findet eine Überschneidung der Arbeitsbereiche des Konstrukteurs und Bauablaufplaners statt, was entsprechende Absprachen der an diesem Prozess beteiligten Personen erfordert. Die Sequenzierung der Vorgänge erfolgt dabei auf Basis von Regeln, deren Erstellung ein entsprechendes Expertenwissen voraussetzt und dadurch vom Anwender kompliziert zu handhaben ist.
Infolge der Abbildung von Bauteilklassen auf konkrete Bauteile des Bauwerksmodells wird in [Morad u. Beliveau 1994] eine Entkopplung der Bauablaufplanung vom CAD-System erreicht. Der Aspekt der Sequenzierung von Vorgängen anhand vom Nutzer in eine Reihenfolge gebrachter geometrischer Zonen wird als aussichtsreich angesehen, da dies der herkömmlichen Arbeitsweise im Sinne einer Einteilung in Arbeitszonen entspricht. Eine Sequenzierung anhand der geometrischen Position von Bauteilen bzw. Bauwerkskomponenten, wie auch in [de Vries u. Hanrik 2007] dargestellt, wird

als problematisch angesehen, da die Reihenfolge der Erstellung beispielsweise nicht zwangsläufig von unten nach oben oder links nach rechts entsprechen muss (vgl. Kapitel 4.1).

Wiederverwendung Durch den Einsatz der Methodik des CBR kann eine Wiederverwendbarkeit von Bauablaufplänen erreicht werden. Voraussetzung dafür ist eine entsprechende Modellierung das Bauablaufs und der darin enthaltenen Vorgänge. Der Aspekt der Übertragbarkeit vorhandener Lösungen auf neue Probleme ist vielversprechend. Insbesondere die automatisierte Zuordnung von Vorgängen zu Bauwerksobjekten, in Kombination mit der Modellierung von Vorgängen mit Voraussetzungen und Resultaten (beispielsweise [Heinrich u. Huhnt 2003] und [Huhnt 2009]), erscheint mithilfe der in [Dzeng u. Whang 2003] beziehungsweise [Mikulakova 2010] dargestellten Ähnlichkeitsbestimmung betrachteter Bauwerksobjekte zielführend.

Unterstützung von Ablaufvarianten Eine Unterstützung von Ablaufvarianten im Prozess der Bauablaufplanung wird von keinem der genannten Forschungsansätze berücksichtigt.

2.2 Entwicklungsbedarf

Bei der gegenwärtig üblichen Vorgehensweise für die Planung eines Bauablaufs erfolgt lediglich eine Dokumentation des Planungsprozesses, wobei sich diese auf das Beschreiben des Endergebnisses beschränkt. Während des Planungsprozesses entscheidet sich der Planer auf Basis der ihm zu diesem Zeitpunkt zur Verfügung stehenden Informationen für genau einen möglichen Bauablauf. Innerhalb des Entscheidungsprozesses zieht er aufgrund seiner Erfahrung mehrere Ausführungsalternativen für das Erreichen von Projektzielen in Betracht. Diese Alternativen gehen bei der herkömmlichen Vorgehensweise verloren und sind im resultierenden Bauablaufplan nicht enthalten. Auch die Entscheidungen, die zu diesem Ergebnis geführt haben, bleiben von der Dokumentation des Bauablaufs ausgeschlossen. Es ist somit bei anschließender Betrachtung des Planungsergebnisses weder möglich festzustellen, warum eine bestimmte Variante gewählt wurde, noch welche möglichen Ausführungsalternativen in Betracht gezogen wurden. Die eigentliche Intention, welche zum konkret geplanten Bauablaufplan geführt hat, geht bei Nutzung herkömmlicher Modelle verloren. Dies muss als problematisch und als großer Nachteil herkömmlicher Modelle angesehen werden. Beispielsweise nimmt sowohl die Qualität als auch die Menge der für die Bauablaufplanung relevanten Informationen mit fortschreitender Planung und während der eigentlichen Bauausführung zu. Es ist somit möglich, dass die Entscheidung des Planers für eine bestimmte Ausführungsalternative zu einem Zeitpunkt getroffen wurde, an dem beispielsweise unzureichende oder auch qualitativ ungenügende Informationen vorlagen. Ebenso ist es möglich, dass sich aufgrund äußerer Bedingungen während des Bauablaufs Randbedingungen ändern, was möglicherweise ebenfalls zu einer anderen Entscheidung geführt hätte. Dieser Umstand führt nicht nur zu eventuell falschen Entscheidungen, sondern auch infolge nicht verfügbarer Alternativen in der Bauablaufplanung

zu verzögerten Reaktionen bei Störungen.

Beispiel: Stahlbetonwände können als Fertigteil oder mittels Ortbeton eingebaut bzw. hergestellt werden. Zu Beginn der Planung wurde festgelegt, dass Stahlbetonwände mittels Ortbeton hergestellt werden. Die Entscheidung wurde aufgrund der innerhalb der ausführenden Firma verfügbaren Technologie und der durch die Herstellung anfallenden Kosten gefällt. Während der eigentlichen Bauausführung kann jedoch plötzlich aufgrund äußerer Störungen der Zeitplan nicht mehr eingehalten werden. Durch eine drohende Konventionalstrafe ist nach den nun vorliegenden Informationen plötzlich die Variante, Wände in Form von Betonfertigteilen einzubauen, die bessere Alternative, da auf diese Weise Zeit eingespart werden kann und die entstehenden Mehrkosten gegenüber der sonst fälligen Konventionalstrafe eine Einsparung bedeuten können.

Weiterhin besteht unter Verwendung herkömmlicher Modelle nur eine begrenzte Möglichkeit der formalen Kontrolle des Bauablaufs. Eine Untersuchung auf Eindeutigkeit und Vollständigkeit des Bauablaufs kann nicht erfolgen. Ebenso ist eine Kontrolle logischer Abhängigkeiten von Teilabläufen nicht oder kaum möglich. Dies ist auf die fehlende Verbindung zwischen dem geplanten Bauablauf und den Informationen, die zur Entwicklung des vorliegenden Bauablaufplans geführt haben, zurückzuführen. Eine Kontrolle des geplanten Bauablaufs erfolgt meist durch aufwändiges Gegenprüfen durch einen anderen Planer oder aber durch wiederholtes Prüfen desselben Planers. Beide Möglichkeiten sind zeitaufwändig, von subjektiven Entscheidungen geprägt und fehlerträchtig.
Ein weiteres Problem ist die fehlende Modellierung von Abhängigkeiten zwischen dem Bauablaufplan und der zugrunde liegenden Bauwerksmodellierung. Eine häufig angewandte Methode, diesem Problem zu begegnen, ist das nachträgliche Modellieren dieser Abhängigkeiten. Als Beispiel ist hier die 4D-Animation des Bauablaufs zu nennen. Der geplante Bauablauf wird unter Verwendung von Softwarewerkzeugen nach dessen Planung mit einem aus der Bauwerksplanung vorhanden Gebäudemodell verknüpft. Dieser Vorgang erfolgt in der Regel mit großem Aufwand händisch. Die so erzeugten Verknüpfungen müssen nach jeder Planungsänderung nachgeführt werden, um auf diese Weise den aktuellen Bauablauf visualisieren zu können. Begründet durch den hohen zeitlichen Aufwand erfolgt die Erstellung einer 4D-Planung in der Regel nur bei großen Projekten. Gleichwohl wird der Nutzen einer solchen Visualisierung gerade bei der Kommunikation des Bauablaufs zwischen den beteiligten Planern und ausführenden Gewerken als hoch angesehen, da sich auf diese Weise die zeitliche Abfolge von Teilabläufen besser nachvollziehen lässt. Ebenfalls wird die Kontrolle logischer Abhängigkeiten erreicht. Dies betrifft beispielsweise die Reihenfolge von Teilabläufen, die dadurch einer visuellen Prüfung unterzogen werden kann. Weiterhin ist die visuelle und zum Teil auch rechnergestützte Kontrolle der Vollständigkeit des Bauablaufplans möglich, indem geprüft wird, ob für alle herzustellenden Bauteile eine entsprechende Verknüpfung zum Bauablaufplan existiert.
Die ungenügende Reaktion auf Änderungen innerhalb der Planung des auszuführenden Bauwerks ist ebenfalls auf die fehlende Verknüpfung zwischen dem geplanten Bauwerk und der geplanten Bauausführung zurückzuführen. Notwendige Anpassungen sind

vom Planer in diesem Fall gleichermaßen händisch in den Bauablaufplan einzupflegen. Bei der Mehrheit von Bauablaufplänen, die dem gegenwärtigen Stand der Technik entsprechend erstellt werden und bei komplexen Bauwerken leicht mehrere tausend Vorgänge enthalten, ist vor allem die Identifikation der dadurch betroffenen Prozesse sehr aufwändig. Herkömmliche Modelle bieten an dieser Stelle keine ausreichende Unterstützung. Häufig erfolgt in der Praxis aufgrund der genannten Probleme eine solche Nachplanung gar nicht oder nur in begrenztem Umfang. Dies hat die Inkonsistenz des geplanten Bauablaufs mit dem realen Ablauf zur Folge, was mit dem Verlust an Kontrolle über den Bauablauf einhergeht.

Ferner muss festgestellt werden, dass geplante und ausgeführte Bauabläufe in ihrer derzeit modellierten Form nicht oder nur im begrenzten Umfang für andere Bauprojekte wiederverwendet werden können. Ein oft angeführter Grund dafür ist, dass Produkte im Bauwesen mit wenigen Ausnahmen als Unikate bezeichnet werden können. Der Bauablauf, der in seiner Gesamtheit zur Herstellung eines Bauwerks führt, leitet sich in großem Maß aus dem geplanten Bauwerk ab. An dieser Stelle ist festzustellen, dass dieser selbst bei identischen Bauwerken an verschiedenen Standorten nicht zwangsläufig wiederholbar ist. Gründe hierfür sind beispielsweise abweichende Baugrundbedingungen oder auch die jeweils vorhandene Infrastruktur, die den Einsatz spezifischer Technologien für das Erstellen des Bauwerks erschweren oder sogar verhindern können. In den meisten Fällen unterscheiden sich Bauwerke jedoch ohnehin infolge ihrer Einmaligkeit, so dass sich die Frage der Wiederverwendbarkeit bereits ausgeführter Bauabläufe meist gar nicht stellt. Zwar wird innerhalb der Bauablaufplanung auf vorhandenes Wissen in Form von existierenden Bauablaufplänen ähnlicher Projekte zurückgegriffen, dies ist jedoch ein manueller, nicht automatisierter Prozess, der von subjektiven Entscheidungen des jeweiligen Planers gekennzeichnet ist. Auch beschränkt sich dieser Vorgang in der Regel nur auf die Übernahme grober Strukturen des existierenden Bauablaufs. Eine Anpassung von Bauablaufplänen bereits umgesetzter Projekte auf die jeweiligen Gegebenheiten reicht in der Regel nicht aus oder ist zu zeitaufwändig. Dies hat häufig die aufwändige Neuerstellung eines Bauablaufplans für jedes aktuell umzusetzende Bauprojekt zur Folge.

2.3 Analyse der Bauablaufplanung

Generell wird ein Bauablauf aus der Art des zu verwirklichenden oder auch umzubauenden Bauwerks abgeleitet. Um eine Realisierung des Bauwerks zu ermöglichen, müssen die für die Verwirklichung des angestrebten Bauwerks notwendigen Informationen eindeutig beschrieben werden. Die Menge dieser Informationen sowie die Art und Weise der Beschreibung wird als *Produktmodell* bezeichnet. Das Modell für die Beschreibung eines Bauablaufs wird als *Prozessmodell* bezeichnet.
Es folgt die Analyse der beiden Modelle sowie die Betrachtung der darin enthaltenen Informationen im Hinblick auf die im Anschluss ebenfalls betrachtete prinzipielle Arbeitsweise bei der Planung eines Bauablaufs. Ziel dieser Untersuchung ist das Ableiten möglicher Lösungsansätze, um die im vorangestellten Kapitel genannten Probleme zu konkretisieren und einen Lösungsansatz zu entwickeln.

2.3.1 Betrachtung des Produktmodells

Die Beschreibung eines Produktmodells im Bauwesen erfolgt auf verschiedene Arten:

Beschreibung durch Baupläne: Bauwerksbeschreibende Informationen liegen dem Planer in der Regel in Form von zweidimensionalen technischen Zeichnungen vor. Die so bezeichneten Baupläne entsprechen der Dokumentation des zum Zeitpunkt der Erstellung vorliegenden Endergebnisses der Bauwerksplanung. Baupläne existieren ausgerichtet auf die zu beschreibenden Informationen sowohl in verschiedenen Detailstufen, als auch Ausrichtungen im Hinblick auf die darin zu beschreibenden Informationen. Ein Gebäude wird dabei in der Regel durch eine Anzahl verschiedener Baupläne, dem Plansatz, beschrieben. Typischer Bestandteil dieser Baupläne ist die Darstellung des Bauwerks in Grundrissen von Struktureinheiten (z.B. Etagen), Schnitte durch das Bauwerk und aus den Bauwerksansichten. Baupläne enthalten sowohl topologische als auch geometrische Informationen. Aus topologischer Sicht betrifft dies die Beschreibung von Beziehungen zwischen Bauteilen - beispielsweise die Zugehörigkeit von Bauteilen zu Gebäudeflügeln oder Etagen ebenso wie die Zuordnung von Fenstern zu Wänden oder deren Eingliederung in definierte Bauabschnitte, die aus den jeweiligen Plänen interpretiert werden müssen. Geometrische Informationen beziehen sich auf die maßhaltige Darstellung von Bauteilen beziehungsweise den Bauwerken hinsichtlich ihrer Geometrie. Baupläne enthalten die Position von Bauteilen innerhalb eines gegebenen Koordinatensystems, sowie deren exakte zweidimensionale geometrische Darstellung in einem angegebenen Maßstab, ergänzt durch die Angabe zusätzlicher Maße, die in der jeweiligen Darstellung nicht ersichtlich sind. Beispiel hierfür sind Brüstungshöhen und die Höhe der Fenster in den Grundrissen. Weiterhin existieren Detailzeichnungen mit einer technisch exakten Beschreibung einzelner Bauteile sowie spezialisierte Pläne wie Schalungs- und Bewehrungspläne. Ebenso geben sie je nach Detaillierungsgrad Auskunft über die zu verwendenden Materialien, die durch entsprechende Schraffuren oder auch Farben dargestellt werden. Der Abriss als auch der Wiederaufbau von Bauteilen kann ebenfalls dargestellt werden, wie es beispielsweise bei der Umnutzung von Bauwerken oder Sanierungsarbeiten notwendig ist. Die Art und Weise der Ausführung dieser Pläne ist nach DIN 1356[14], ISO 4157[15], ISO 2594[16], ISO 7518[17] und weiteren Vorschriften und Arbeitsrichtlinien genormt.

Ergänzt werden diese Pläne durch weitere Informationen, die in unmittelbarem Zusammenhang mit der Beschreibung des geplanten Bauwerks stehen. Diese Informationen sind in zusätzlichen, die Planung begleitenden Dokumenten enthalten. Sie beinhalten zum einen Teil ausgewertete Informationen, die in den Plänen zwar enthalten, aber nicht unmittelbar ersichtlich sind, wie beispielsweise Bauteil-, Material- und Bemusterungslisten. Zum anderen handelt es sich um zusätzliche Informationen, die beispielsweise die technische Ausrüstung eines Bauwerks im Detail beschreiben.

[14]DIN 1356 Bauzeichnungen
[15]ISO 4157 Zeichnungen für das Bauwesen
[16]ISO 2594 Bauzeichnungen; Projektionsmethoden
[17]ISO 7518 Zeichnungen für das Bauwesen; Vereinfachte Darstellung von Abriss und Wiederaufbau

Auch diese zusätzlichen Informationen können mittels der Festlegungen nach DIN EN 61355 Teil C[18] eindeutig abgebildet werden.

Beschreibung durch Bauwerksinformationsmodelle: Zunehmende Bedeutung erlangen digitale Bauwerksinformationsmodelle aufgrund ihrer erweiterten Möglichkeiten. Der Informationsgehalt ist dabei abhängig von der Art und dem möglichen Detaillierungsgrad der angewendeten Modellierung. Modelle wie beispielsweise die IFC, die eine allgemeingültige Abbildung für alle Bereiche des Bauwesens anstreben, versuchen die Abbildung aller für das Bauwesen notwendigen Informationen innerhalb eines Modells. Solche Modelle weisen eine sehr hohe Informationsdichte auf.
Geometrische, topologische als auch das Bauwerk weiterführend beschreibende Informationen werden innerhalb einer Datenstruktur abgebildet. In der Regel folgt die Abbildung einer solchen Datenstruktur dem objektorientierten Ansatz. Ein Bauwerk wird dabei mittels seiner Bauteile beschrieben. Diese werden als Objekte mit ihren zugrunde liegenden Eigenschaften abgebildet. Beispielsweise wird eine Wand auch als ein solches Objekt abgebildet und muss nicht infolge ihrer geometrischen Darstellung - wie beispielsweise vier Linien und einer Schraffur - interpretiert werden. Vielmehr ist die geometrische Repräsentation als eine Eigenschaft dieser Wand anzusehen, welche gleichzeitig auch deren geometrische Abmessungen beinhaltet. Die geometrische Beschreibung erfolgt dabei in der Regel dreidimensional und unter Verwendung der realen Maße. Gleichwohl werden in solchen Modellen mehrere Darstellungsarten wie zum Beispiel auch die zweidimensionale Repräsentation für Pläne des vorangegangenen Abschnitts unterstützt. Topologische Strukturen eines Gebäudes oder auch Beziehungen von Bauteilen untereinander werden durch Relationen zwischen den jeweiligen Objekten abgebildet. Sie bedürfen keiner Interpretation, sondern können direkt abgerufen werden. Dies betrifft beispielsweise die Zugehörigkeit von Bauteilen zu Etagen oder Räumen. Auch die Zuweisung von Materialien oder anderen weiterführenden Informationen wird über Relationen abgebildet.
Ein Großteil der in der Praxis eingesetzten Software bildet intern eine solche Datenstruktur ab. Auch die Vorgehensweise bei der Planung eines Bauwerks ist zum Teil an diese Struktur angelehnt, indem das Bauwerk ausgehend von Struktureinheiten und Bauteilen geplant wird (Graphisoft ArchiCAD, Autodesk Revit, Nemetschek Allplan und weitere). Eine nicht proprietäre Exportmöglichkeit des in der Planungssoftware vorliegenden Bauwerksinformationsmodells ist in der Regel als STEP-basierte IFC-Datei möglich.

2.3.2 Analyse des Prozessmodells

Das Prozessmodell im Bauwesen bildet die Menge der notwendigen Informationen ab, um den für ein geplantes Bauvorhaben angestrebten Bauablauf hinreichend genau zu beschreiben. Es beschreibt alle notwendigen Arbeitsschritte, die in ihrer technologisch

[18] DIN EN 61355 Klassifikation und Kennzeichnung von Dokumenten für Anlagen, Systeme und Einrichtungen - Teil C für den technischen Bereich des Bauwesens. Die Norm regelt die einheitliche und Hersteller übergreifende Klassifikation und Identifikation von Dokumenten. Sie ist international gleich unter der Bezeichnung IEC 61355

richtigen Reihenfolge zur Realisierung der angestrebten Baumaßnahme führen. Ein solcher Arbeitsschritt wird als Vorgang oder auch Prozess bezeichnet - z.B. *„Rohbau Erdgeschoss“* oder *„Erstellung Außenwände 1.-OG“*. Dabei ist der Inhalt des Bauablaufplans nicht auf die Bauausführung selbst beschränkt. Es werden auch Vorgänge hinsichtlich der Ausschreibung und Vergabe von Leistungen, der Beschaffung von Material, der Baustelleneinrichtung, der Bauabnahme und der Inbetriebnahme des Bauwerks berücksichtigt.
Ziel der Bauablaufplanung ist die terminliche Koordination des Bauablaufs. Vorgänge werden somit in ihrem zeitlichen Zusammenhang betrachtet. Jedem Vorgang ist eine Dauer zugeordnet, die die notwendige Zeit beschreibt, um diesen Vorgang auszuführen. Ebenso ist eine Zuordnung anderer für den Bauablauf relevanter Größen, wie beispielsweise Kosten oder Ressourcen, zu Vorgängen möglich.
Ein weiterer Inhalt des Prozessmodells ist die Abbildung von Abhängigkeiten, die zwischen den Vorgängen des Bauablaufs existieren. Dies betrifft die schon erwähnte Reihenfolge von Vorgängen.
Die Dokumentation der Terminplanung erfolgt je nach gewünschtem Detailgrad durch verschiedene graphische Darstellungsarten wie Terminliste, Gantt-Diagramm[19], Liniendiagramm (bei Linienbaustellen, z.B. im Straßen- oder Tunnelbau) und Netzplan. Alle Dokumentationsformen finden Verwendung, wobei Balkenpläne in der Praxis am häufigsten für die Darstellung des Bauablaufs genutzt werden.
Die Dokumentation des Bauablaufs ist auch unter Verwendung der IFC möglich, inklusive der Zuordnung von Vorgängen zu Bauteilen.

2.3.3 Analyse des Planungsprozesses

Bauablaufpläne existieren in unterschiedlichen Detaillierungsstufen und Abbildungen. Bei der Planung von Bauabläufen werden verschiedene Phasen durchlaufen, in denen der Bauablaufplan in verschiedenen Detailstufen entwickelt wird (die Analyse der Bauablaufplanung erfolgt in Anlehnung an [Bauer 2007]).
Ausgehend von einer Rahmen- oder Grobterminplanung erfolgt durch eine Arbeitspaketzerlegung (WBS[20]) die schrittweise Verfeinerung, die bis zur Zerlegung in für das Projekt relevante Einzelaufgaben führt. Man bezeichnet die weiteren beiden Phasen als Koordinations- oder Steuerungsplan und Feinterminplan. Die einzelnen Phasen und deren Inhalte sind in Abbildung 2.1 ersichtlich. Als Grundlage der Planung dienen hierfür die Ausführungspläne und das Leistungsverzeichnis sowie vorhandene Informationen über Randbedingungen wie die Qualität der Ausführung oder geforderte Termine.
Innerhalb des Grobterminplans erfolgt zunächst die Erstellung einer hierarchischen Gliederungsstruktur, die zum Teil von der Art des Bauwerks abhängig ist. Üblicherweise erfolgt die Gliederung dabei nach Bauteilen, Geschossen, Bauabschnitten, den Ausführungsphasen, den Gewerken sowie einer Einteilung in Teilbauleistungen. Kann eine Gliederung aufgrund des Bauwerkscharakters nicht erfolgen, wird eine entsprechende Einteilung in Bauabschnitte vorgenommen. Häufig werden diese groben Strukturen aus

[19] graphische Darstellung der zeitlichen Abfolge von Vorgängen
[20] engl.: Work Breakdown Structure

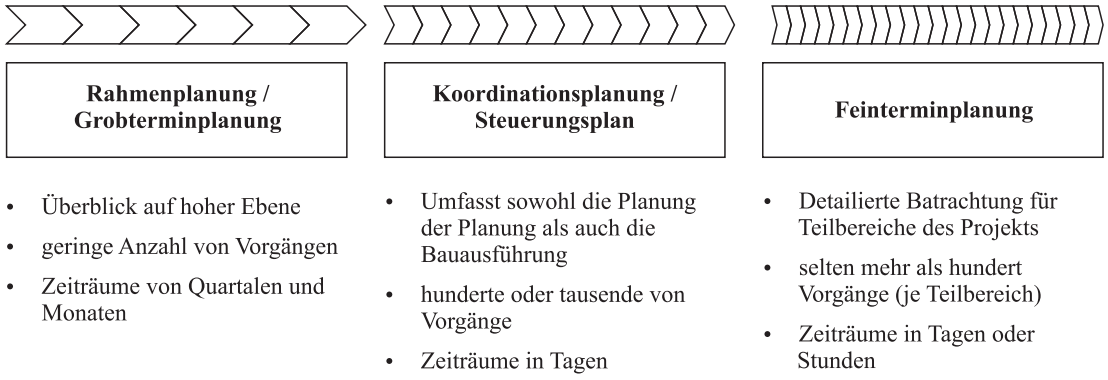


Abbildung 2.1: Phasen der Prozessmodellierung

früheren Planungen für ähnliche Bauwerke übernommen. Eine Anpassung ist jedoch in der Regel selbst bei dieser groben Strukturierung notwendig.
Während der Erstellung des Koordinationsplans erfolgt die Festlegung der einzusetzenden Bauverfahren und somit der auszuführenden Vorgänge. An dieser Stelle wird eine subjektive Entscheidung des Planers für ein bestimmtes Bauverfahren getroffen. Gewöhnlich basiert diese Auswahl auf Erfahrungswerten des jeweiligen Planers. Es findet somit die Selektion einer möglichen Ausführungsvariante statt, die in Form der gewählten Vorgangsabfolge im Bauablauf festgeschrieben wird.
Nachfolgend wird die technologisch richtige Reihenfolge der Vorgänge bestimmt, aus der sich eine Ablaufstruktur ergibt. Dabei werden Vorgänge in die zuvor entwickelte Gliederungsstruktur eingeordnet, wodurch eine erste grobe Reihenfolge der Vorgänge entsteht. Die weitere Anordnung von Vorgängen und deren Definition ist abhängig von der Tiefe der Planung - ihrem Detaillierungsgrad. Die Detailstufe der Vorgänge ist dabei abhängig vom Fortschritt der Planung und ihrer erforderlichen Qualität. In [Bauer 2007] wird für die Bezeichnung der Detailstufen von Ablaufschritten die in Abbildung 2.2 gezeigte Einteilung vorgenommen. Häufig bricht die Planung hier schon auf einer Stufe zwischen dem Teilablauf und der Ablaufstufe ab. Die Feinplanung findet dann meist während der Bauphase auf der Baustelle statt.
Nach der Erstellung des prinzipiellen Ablaufs erfolgt die Zuordnung der Arbeitsdauer zu den Vorgängen. Die Dauer von Vorgängen wird in der Regel durch den Planer auf Basis von Aufwands- oder Leistungswerten berechnet oder beruhen auf Erfahrungswerten. Ausgehend von einem Startdatum für einen Vorgang, der sich aus dem Enddatum des vorangestellten Vorgangs ergibt, bestimmt sich, wann ein Vorgang beginnt und wann er endet. In diesem Zusammenhang spricht man ab dieser Stufe der Bauablaufplanung von einem *Terminplan*.
Es folgen abschließende weitere Schritte der Planung, wie das Erstellen von Arbeitsverzeichnissen, die Planung der Baustelleneinrichtung sowie die Leistungs- und Kapazitätsabstimmung für die Einhaltung vorgegebener Termine.
Alle oder auch nur Teile der genannten Schritte sind mitunter wiederholt auszuführen. Dies ist der Fall, wenn sich Informationen ändern, auf deren Basis Entscheidungen getroffen wurden, die zum geplanten Bauablauf geführt haben. Beispielsweise trifft dies in hohem Maße auf Informationen zu, die aus der Bauwerksplanung gewonnen werden.

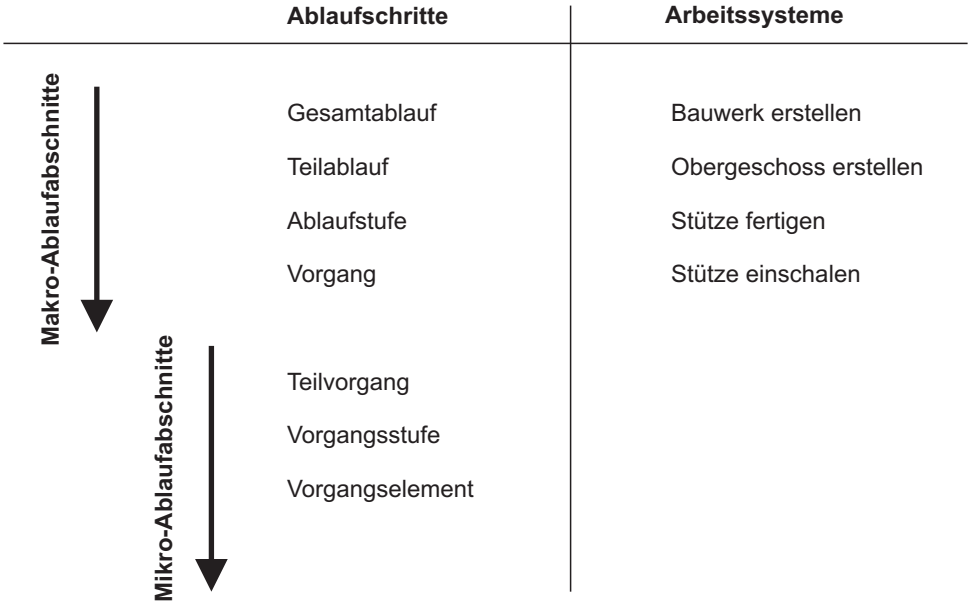


Abbildung 2.2: Gliederung des Arbeitssystems [Bauer 2007]

Auch während der Bauausführungsphase, in welcher der Terminplan der Kontrolle des Bauprozesses dient, kann die Nachführung des Terminplans aufgrund von Störungen notwendig werden.

2.4 Problembetrachtung

Innerhalb der vorangestellten Kapitel wurde die prinzipielle Vorgehensweise der Bauablaufplanung und die dafür zugrunde liegenden Informationen betrachtet. Dabei konnten verschiedene Probleme derzeitig angewendeter Methoden und Modelle für das Erstellen einer Bauablaufplanung erkannt werden. Es werden zusammenfassend folgende Kernprobleme festgestellt, auf die innerhalb dieser Arbeit eingegangen werden soll:

Integration von Planungsalternativen: Derzeitige Modelle dokumentieren jeweils nur das Endergebnis der Planung. Die eigentliche Intention, warum ein Bauablauf in der vorliegenden Form entwickelt wurde, geht dabei verloren. Ein neues Modell muss die Planungsintention mit einbeziehen. Prinzipiell handelt es sich hierbei im weitesten Sinn um die Abbildung von Wissen unter verschiedenen Gesichtspunkten. Vom Planer betrachtete Planungsalternativen sollen dabei in die Modellierung des Bauablaufs einfließen. Dies gewährleistet die mögliche Gegenüberstellung dieser Alternativen unter verschiedenen Gesichtspunkten. Weiterhin wird hiermit nicht nur die Voraussetzung geschaffen, Entscheidungen rückgängig zu machen, sondern auch deren Dokumentation abzubilden, was die Beantwortung folgender Fragen ermöglicht:

- Warum wurde eine bestimmte Variante gewählt?
- Was hat den Planer dazu bewegt, diese Entscheidung zu treffen?

- Welche Alternativen wurden für die Entscheidungsfindung überhaupt betrachtet?
- Gibt es weitere Alternativen, die noch gar nicht betrachtet wurden?
- Ist eine Alternative aufgrund aktueller Informationen besser geeignet?
- Kann aufgrund einer Störung im Bauablauf eventuell mit einer Variante geantwortet werden?

Erzeugen konsistenter Bauabläufe: Die Nachführung geplanter Bauabläufe bei einer geänderten Bauwerksplanung wie auch während der eigentlichen Bauausführung ist sehr aufwändig. Sie wird durch existierende Werkzeuge für die Bauablaufplanung infolge der in der Regel fehlenden Verknüpfung zwischen dem Bauwerksmodell und dem Prozessmodell nicht ausreichend unterstützt. Dies hat häufig Inkonsistenzen zwischen dem Bauwerksmodell und dem Prozessmodell zur Folge. Dieser Informationsverlust muss durch das zu entwickelnde Modell minimiert werden. Es muss demzufolge die logischen Abhängigkeiten zwischen beiden Modellen abbilden können. Eine Unterstützung des Planers mittels entsprechender Werkzeuge ist dabei von großer Bedeutung. Ziel ist eine teilautomatisierte Abbildung der logischen Zusammenhänge zwischen Produkt- und Prozessmodell, um dem Planer die Konsistenzsicherung zwischen beiden Modellen zu ermöglichen.

Wiederverwendbarkeit der Bauablaufplanung: Eine effektive Wiederverwendbarkeit der Bauablaufplanung ist bei Verwendung derzeitiger Modelle äußerst begrenzt oder gar nicht möglich. Es erfolgt lediglich die Übernahme grober Strukturen und Teilabläufe. Um diesem Umstand zu begegnen, muss der Bauablauf in einer Form modelliert und persistent gespeichert werden können, die eine Wiederverwendbarkeit von Bauablaufplänen ermöglicht. Ziel ist dabei eine automatisierte Erzeugung von Bauablaufplänen auf Basis dieser gespeicherten Informationen.

Integration der visuellen Kontrolle: Die Vorteile einer 4D-Planung sind im Allgemeinen anerkannt. So hat diese Form der Planung bereits Einzug in die praktische Anwendung gehalten. Die Problematik besteht in der Erstellung solcher Modelle. Derzeitig erfolgt die Verknüpfung des Bauablaufs mit dem virtuellen Bauwerk, nachdem der Bauablauf fertig geplant wurde. Eine vollständige Integration in den Prozess der Bauablaufplanung existiert nicht. Da die Erstellung und Nachpflege eines solchen Modells in der Regel händisch erfolgt, ist der Aufwand für die Erstellung einer solchen Planung sehr hoch. Eine 4D-Planung erfolgt dadurch meist nur bei größeren Projekten, die diesen Aufwand rechtfertigen. Auch an dieser Stelle ergibt sich das Problem der Inkonsistenz zwischen den beiden Modellen und erfordert die bessere Unterstützung des Planers durch Entwicklung entsprechender Werkzeuge. Ziel des neuen Modells ist die Integration der 4D-Planung in die Bauablaufplanung, um eine konsistente 4D-Planung zu ermöglichen, sowie die Vorteile einer solchen Planung für kleinere Projekte zu ermöglichen und den Aufwand für große Projekte zu minimieren.

2.5 Lösungsstrategie

Ausgehend von den Schlussfolgerungen des letzten Kapitels wird eine generelle Lösungsstrategie entwickelt, die zur Modellbildung in Kapitel 4 führt.

2.5.1 Bestimmung von Vorgängen

Bei der Erstellung von Bauablaufplänen besteht eine starke Abhängigkeit zum geplanten Bauwerk. Das Bauwerk selbst wird durch das Produktmodell beschrieben. Mit der Verfügbarkeit von standardisierten Bauwerksinformationsmodellen (BIM), beispielsweise auf Basis der IFC, liegen diese Informationen in einer definierten, automatisiert auswertbaren Datenstruktur vor, die unter Verwendung aktueller Planungswerkzeuge im Zuge der Bauwerksplanung erstellt werden kann.

Grobterminplanung: Die grobe Struktur eines Bauablaufplans kann aus den im BIM enthaltenen Informationen abgeleitet werden, da die analog zur etablierten Vorgehensweise benötigten Informationen bezüglich der Bauwerksstruktur darin enthalten sind. Dies entspricht im Sinne von Abbildung 2.2 der Definition von Teilabläufen - beispielsweise auf Geschoss- oder Bauabschnittebene - und entspricht der Grobterminplanung, welche die erste Phase der Prozessmodellierung darstellt.
Die teilautomatisierte Zuordnung von Bauteilen für die Definition von Ablaufstufen bis hin zu Vorgängen und zu den Teilabläufen der Grobterminplanung kann ebenfalls auf Basis der im BIM vorhandenen Informationen erfolgen. Sämtliche Bauteile wie Fundamente, Wände oder Stützen liegen in einer objektorientierten Form vor und sind durch auswertbare Relationen im BIM in die Gebäudestruktur eingebettet. Dies betrifft beispielsweise die Zuordnung von Bauteilen zu Etagen, Räumen oder auch Bauabschnitten.

Feinterminplanung: Infolge einer Zuordnung von Materialien zu Bauteilen im BIM ist der Rückschluss auf die für das Bauteil anzuwendenden Bauverfahren realisierbar und ermöglicht somit die Ableitung von Vorgängen im Sinne einer Feinterminplanung. Bedingung hierfür ist die Abbildung und Speicherung der Bauverfahren in einer Form, die einen solchen Rückschluss formal zulässt. Gleichwohl existiert an dieser Stelle die Integrationsmöglichkeit von Varianten im Modell, da verschiedene Möglichkeiten für die Herstellung von Bauteilen infolge der Existenz verschiedener Technologien für deren Herstellung existieren. Das Modell darf daher nicht auf die Beschreibung von nur einer möglichen Technologie für ein angestrebtes Teilziel beschränkt sein.

2.5.2 Übertragbarkeit von Vorgängen und Bestimmung der Reihenfolge

Gemeinsamkeiten verschiedener Bauwerke: Betrachtet man Bauwerke auf der Detailebene eines BIM, wird deutlich, dass ein Bauwerk aus einer Menge von Bauteilen wie beispielsweise Fundamenten, Stützen oder Wänden besteht. Weiterhin existiert für ein Bauwerk eine topologische Struktur. Ein Bauwerk wird in der Regel in Abschnitte eingeteilt, denen wiederum Bauteile zugeordnet sind. Die topologische Struktur von

Bauwerken ist dabei meist übertragbar, die darunter liegende Geometrie jedoch in der Regel nicht. Auf dieser Detailebene wird deutlich, dass es sich in ihrer Gesamtheit zwar aufgrund ihrer abweichenden Geometrie um verschiedene Gebäude handelt, diese jedoch aufgrund ihrer Bestandteile ähnlich sein können.

Gemeinsamkeiten verschiedener Bauablaufpläne: Wendet man die Aussage des letzten Absatzes analog auf einen Bauablaufplan an, führt dies zum gleichen Ergebnis. Zerlegt man einen Bauablaufplan in dessen Bestandteile, so wird deutlich, dass dieser aus einer Menge von Vorgängen besteht, die in technologisch richtiger Reihenfolge ausgeführt das Bauwerk als solches erstellen helfen. Ein Vorgang oder auch eine Kette von Vorgängen ändert dabei entweder den Zustand eines Bauteils oder führt zur Erstellung eines neuen Bauteils mit bestimmten Eigenschaften. Da eine Kette von Vorgängen auch durch einen einzelnen Vorgang infolge eines gröberen Detaillierungsgrades substituiert werden kann, soll an dieser Stelle vereinfacht gelten, dass ein Vorgang zur Erstellung eines Bauteils mit definierten Eigenschaften führt (siehe Abbildung 2.3).

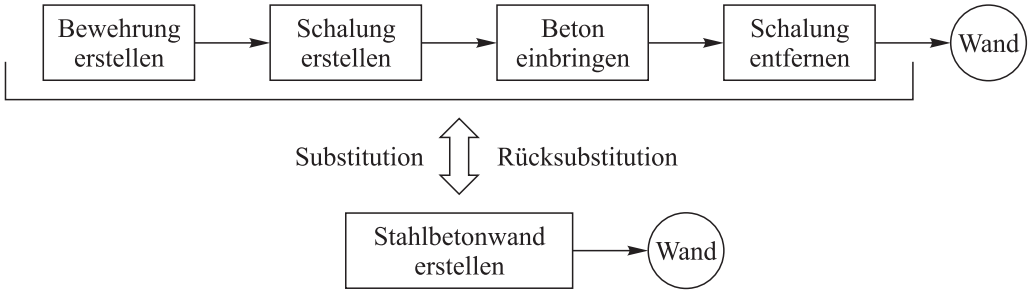


Abbildung 2.3: Beispiel für die Substitution von Vorgangssketten

Schlussfolgerung: Führt man die Aussagen der letzten beiden Abschnitte zusammen, lassen sich daraus folgende drei Thesen ableiten:

These 1 *Ein Vorgang resultiert technologisch in einem Bauteil, das sich zum betrachteten Zeitpunkt in einem definierten Zustand befindet. Das Bauteil und dessen Zustand kann in Form einer Bauteilbeschreibung formal abgebildet werden.*

These 2 *Bauteile können miteinander hinsichtlich ihres Typs und ihrer Eigenschaften verglichen werden. Stimmen deren Typ und Eigenschaften in hohem Maß überein, so sind die betrachteten Bauteile zueinander ähnlich. Haben zwei einander gegenübergestellte Bauwerke ähnliche Bauteile, so sind die zur Erstellung der Bauteile notwendigen Vorgänge übertragbar. Sie stellen zumindest eine mögliche Variante für die Erstellung des betrachteten Bauteils dar.*

These 3 *Vergleicht man Bauteile eines Bauwerkes, für dessen Erstellung ein Bauablaufplan erzeugt werden soll, mit einem Bauwerk, für das bereits Vorgänge definiert wurden, so kann bei ähnlichen Bauteilen ein wahrscheinlicher Vorgang als eine mögliche Variante für deren Erstellung bestimmt werden. Als notwendige Voraussetzung müssen alle Vorgänge mit ihren korrespondierenden Bauteilen im bestehenden Bauablaufplan oder zumindest mit einem, das jeweilige Bauteil für einen Vergleich hinreichend genau beschreibenden, Datensatz verknüpft sein.*

Bestimmung der Reihenfolge: Ein Bauablaufplan hat einen Startpunkt, an dem die Erstellung des Bauwerks beginnt, und einen Endpunkt, an dem die Erstellung des Bauwerks abgeschlossen ist. Zwischen Start- und Endpunkt werden die Vorgänge in einer sequentiellen Reihenfolge abgearbeitet, die sich aus der technologisch richtigen Abfolge der Vorgänge ergibt. Ein Fenster kann in zeitlicher Abfolge beispielsweise erst dann eingebaut werden, wenn die zugehörige Wand bereits existiert. Ein weiteres Beispiel ist das Einbauen eines Riegels zum Komplettieren eines Rahmens, der erst eingebaut werden kann, wenn seine zugehörigen Stützen bereits errichtet wurden (siehe Abbildung 2.4).

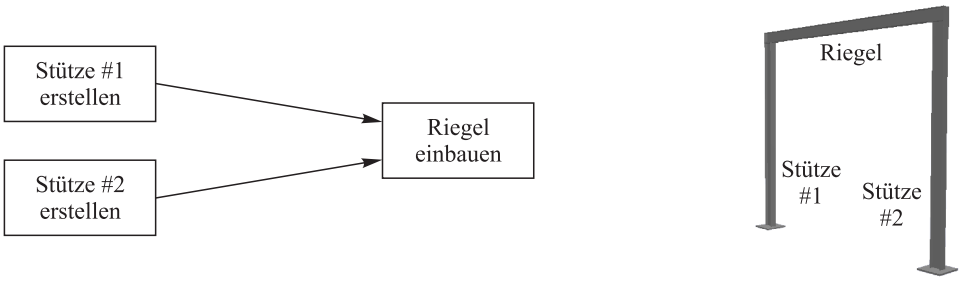


Abbildung 2.4: Abhängigkeiten von Vorgängen am Beispiel eines Rahmens

Demnach existieren zwischen Vorgängen logische Abhängigkeiten und es gelten folgende beiden Aussagen:

Vorgängerbeziehung: Ein Vorgang kann nur dann zur Ausführung kommen, wenn seine Vorgänger ausgeführt wurden und somit seine notwendigen Voraussetzungen hinreichend erfüllt sind.

Nachfolgerbeziehung: Ein Vorgang stellt die notwendige Voraussetzung für seine Nachfolger dar, die nur dann ausgeführt werden können, wenn er selbst ausgeführt wurde.

Da nach These 1 Vorgänge als Ergebnis Bauteile in einem definierten Zustand liefern, muss dieses Bauteil die Voraussetzung für alle nachfolgenden Vorgänge darstellen. Das durch den nachfolgenden Vorgang erzeugte Bauteil ist wiederum die notwendige Voraussetzung für alle Nachfolger dieses Vorgangs. Bauteile können somit als technologisch notwendige Bedingungen für das Ausführen von Vorgängen angesehen werden. Es lassen sich daraus folgende Thesen ableiten:

These 4 *Für einen Vorgang können Voraussetzungen und Resultate definiert werden. Sowohl die Voraussetzungen als auch die Resultate eines Vorgangs lassen sich durch Bauteile in einem definierten Zustand beschreiben.*

These 5 *Durch die Verknüpfung der Resultate eines Vorgangs mit identischen Vorbedingungen anderer Vorgänge kann die logische Reihenfolge von Vorgängen hergestellt werden.*

These 4 und 5 werden in Abbildung 2.5 verdeutlicht. Durch die Kopplung der Bauteile *Stütze#1* und *Stütze#2* wird die logische Reihenfolge der Vorgänge *Stütze erstellen* und *Riegel einbauen* hergestellt.

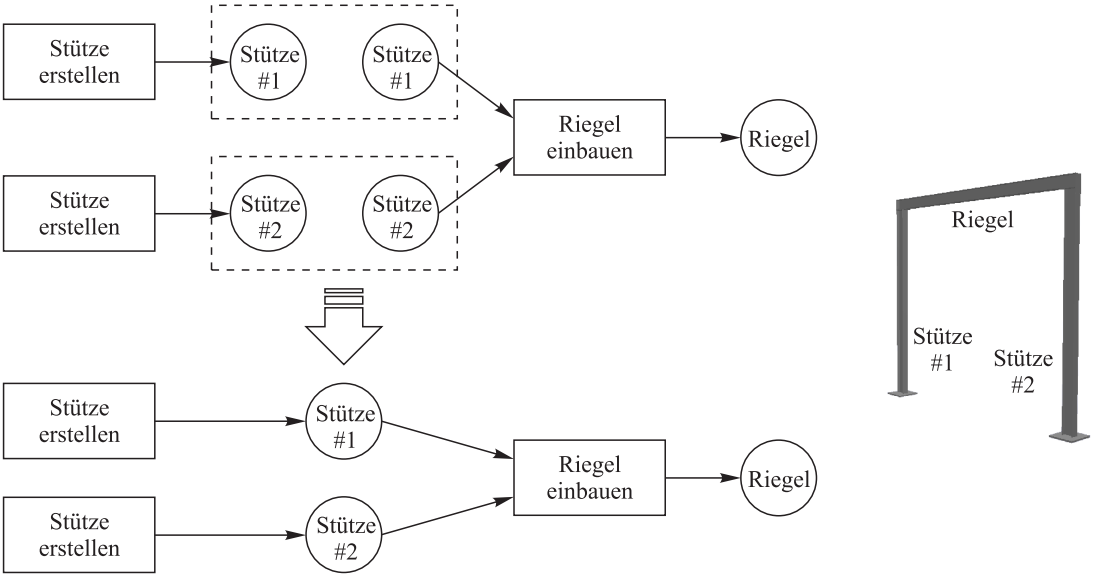


Abbildung 2.5: Zusammenhang von Prozessen

2.5.3 Automatisierungsansatz

Aus den Thesen des vorangestellten Kapitels 2.5.2 lässt sich ein Ansatz für eine teilautomatisierte Generierung von Bauablaufplänen ableiten.

Abbildung von Vorgängen mit Voraussetzungen und Resultaten: Der Grundgedanke besteht zunächst darin, den gesamten Bauablauf auf Basis eines jeden einzelnen Vorgangs mit den für dessen Ausführung notwendigen Voraussetzungen und dessen nach seiner Ausführung erzeugtem Resultat formal abzubilden. Ein ähnlicher Ansatz wurde bereits in [Heinrich u. Huhnt 2003] und in den darauf folgenden Arbeiten [Huhnt 2005], [Enge 2005] sowie [Enge 2010] verfolgt.
Die Beschreibung von Voraussetzungen und des Resultats eines Vorgangs erfolgt dabei in Form von Bauteilen, die sich für den gewählten Zeitpunkt der Abbildung in einem definierten Zustand befinden. Dieser Zustand oder auch Status eines Bauteils wird durch seine für den Abbildungszeitpunkt bestehenden Eigenschaften ausgedrückt.
Voraussetzungen und Resultat eines Vorgangs lassen sich anhand eines *logischen Datensatzes*[21] formal beschreiben. Ziel dieser Vorgehensweise ist die Entkopplung der Vorgänge vom konkreten Bauprojekt. Es ermöglicht die voneinander unabhängige Definition von Vorgängen und erlaubt gleichzeitig die projektunabhängige Speicherung von bauteilbezogenen Prozessinformationen. Es werden folgende Definitionen getroffen:

Definition 1 *Voraussetzungen eines Vorgangs werden jeweils durch einen logischen Datensatz abgebildet, der ein Bauteil mittels seiner Eigenschaften beschreibt. Der Abbildungszeitpunkt für die Eigenschaften von Voraussetzungen eines Vorgangs ist der unmittelbare Zeitpunkt, bevor dieser ausgeführt werden kann.*

[21] vgl. [Claus u. Schwill 2003] Ein Datensatz ist die kleinste logisch in sich abgeschlossene Datenmenge - hier für die Beschreibung eines Bauteils

Definition 2 *Das Resultat eines Vorgangs wird durch einen logischen Datensatz abgebildet, der ein Bauteil mittels seiner Eigenschaften beschreibt. Der Abbildungszeitpunkt für die Eigenschaften eines Resultats ist der unmittelbare Zeitpunkt, nachdem dieser ausgeführt wurde.*

3 Grundlagen und Begriffe

Das vorliegende Kapitel enthält notwendige Grundlagen und definiert Begriffe, die zum Verständnis der Arbeit dienen. Es behandelt Methoden und Algorithmen auf bipartiten Graphen und Workflowgraphen, die im weiteren Verlauf dieser Arbeit für das Modell zum Generieren von Bauablaufplänen herangezogen werden.
Das Kapitel beschränkt sich dabei auf das Minimum an notwendigen Informationen, die für das Verständnis der Arbeit einerseits als auch für die Entwicklung von Algorithmen in den nachfolgenden Kapiteln andererseits notwendig sind. Die formale Beschreibung basiert dabei vorwiegend auf [Pahl u. Damrath 2000].

3.1 Bipartite Graphen

Bipartite Graphen eignen sich, um Beziehungen zwischen zwei voneinander verschiedenen Mengen formal zu beschreiben. Dabei dürfen jeweils nur Kanten existieren, die eine Abbildung zwischen diesen Mengen repräsentieren. Ein solcher Graph besteht aus jeweils zwei voneinander disjunkten[1], endlichen Knoten- und Kantenmengen. Nachfolgende Definitionen basieren auf [Pahl u. Damrath 2000] und [König 2004]).

Definition: Ein Graph $G = (V_1, V_2; R_1, R_2)$ heißt bipartiter Graph, wenn V_1, V_2 disjunkte Knotenmengen und R_1, R_2 disjunkte Kantenmengen mit $R_1 \subseteq V_1 \times V_2$ und $R_2 \subseteq V_2 \times V_1$ sind. Eine Kante von Knoten $x \in V_1$ zu Knoten $y \in V_2$ wird mit dem geordneten Paar $(x, y) \in R_1$ bezeichnet, eine Kante von Knoten $y \in V_2$ zu Knoten $x \in V_1$ mit dem geordneten Paar $(y, x) \in R_2$.

$$G := (V_1, V_2; R_1, R_2) \quad R_1 \subseteq V_1 \times V_2 \quad R_2 \subseteq V_2 \times V_1 \tag{3.1}$$

V_1, V_2	Disjunkte Menge der Knoten
R_1, R_2	Disjunkte Menge der geordneten Knotenpaare (Kanten)

Abbildung 3.1 zeigt als Beispiel die graphische Darstellung eines bipartiten Graphen.

[1] Zwei Mengen sind voneinander disjunkt, wenn kein Element der einen Menge ein Element der anderen Menge ist.

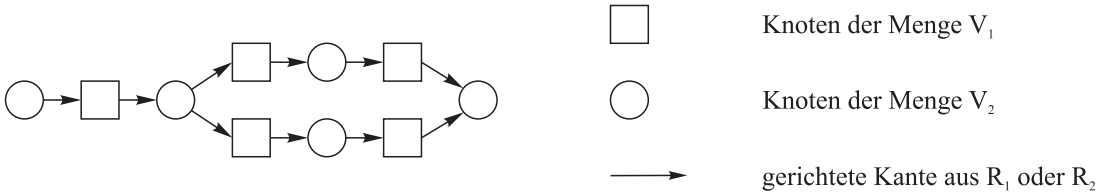


Abbildung 3.1: Beispiel für einen bipartiten Graphen G

Vorgänger und Nachfolger: Ein Knoten y besitzt dann einen Vorgänger x, wenn eine Kante $r(x,y) \in R_1 \mid x \in V_1 \wedge y \in V_2$ oder $r(x,y) \in R_2 \mid x \in V_2 \wedge y \in V_1$ existiert. Die Menge der Vorgänger von x wird mit $t_V(x)$ bezeichnet. Ein Knoten y ist Nachfolger von x, wenn eine Kante $r(x,y) \in R_1 \mid x \in V_1 \wedge y \in V_2$ oder $r(x,y) \in R_2 \mid x \in V_2 \wedge y \in V_1$ existiert. Die Menge der Nachfolger von x wird mit $t_N(x)$ bezeichnet. Die Anzahl der Vorgänger von x wird mit $g_V(x) = |t_V(x)|$, die Anzahl der Nachfolger mit $g_N(x) = |t_N(x)|$ bezeichnet. Die Anzahl der Vorgänger und Nachfolger wird auch Vorgängergrad und Nachfolgergrad genannt.

Vorgänger von x:	$t_V(x) = Rx$
Nachfolger von x:	$t_N(x) = R^T x$
Vorgängergrad von x:	$g_V(x) = \|t_V(x)\|$
Nachfolgergrad von x:	$g_N(x) = \|t_N(x)\|$

3.2 Algorithmen und Methoden auf bipartiten Graphen

Dieses Kapitel beschreibt Methoden und Algorithmen auf bipartiten Graphen, die für die Intention dieser Arbeit zielführend sind. Für eine nachvollziehbare Beschreibung der Algorithmen wird, soweit nicht anders angegeben, der nachfolgende Graph zugrunde gelegt:

$$G := (T, B; P, R) \qquad P \subseteq B \times T \qquad R \subseteq T \times B \tag{3.2}$$

$$B := \{1,2,3,4,5,6\} \quad T := \{a,b,c,d,e\}$$
$$P := \{(1,a),(2,b),(3,c),(3,d),(4,e),(5,e)\}$$
$$R := \{(a,2),(a,3),(b,4),(c,5),(d,5),(e,6)\}$$

Der Graph besteht aus der Menge der Knoten B und der Menge der Knoten T. Zwischen den Knoten existiert die Kantenmenge $P = \{(b,t) \in B \times T\}$ und die Kantenmenge $R = \{(t,b) \in T \times B\}$ (siehe Abbildung 3.2).

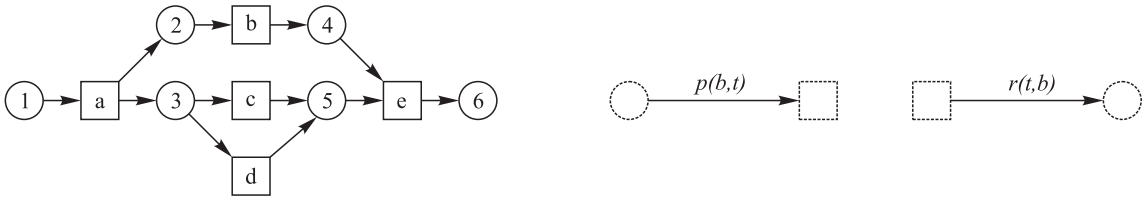


Abbildung 3.2: Beispielgraph G für die Darstellung von Algorithmen auf bipartiten Graphen

3.2.1 Identifikation von Start- und Endknoten

Start- und Endknoten eines Graphen können durch Betrachtung der Struktur des Graphen bestimmt werden.

Startknoten

Als Startknoten werden alle Knoten $b \in B$ oder $t \in T$ bezeichnet, die keine hinführenden Kanten haben. Oder anders ausgedrückt, alle Knoten $b \in B$ oder $t \in T$, die keinen Vorgänger haben, sind Startknoten.

$$startNodes := \{sk_i \in \{B \cup T\} \mid g_V(sk_i) = 0\} \tag{3.3}$$

Algorithmus 1 bestimmt nach Formel 3.3 die Menge der Startknoten. Als Eingangswert dient der auf das Vorhandensein von Startknoten zu untersuchende bipartite Graph G. Als Ergebnis wird die Menge der identifizierten Startknoten zurückgeliefert.

Algorithmus 1: getStartNodes(G)
Bestimmung der Menge von Startknoten

input : BipartiteGraph $G := (T, B; P, R)$
output : **return** Menge der Startknoten *startNodes*

```
1 startNodes := {B ∪ T};
2 foreach Edge e_i(f, s) ∈ {P ∪ R} do
3 |   startNodes := startNodes \ {s} ;      // trage s aus der Menge der
  |   Startknoten aus
4 end
5 return startNodes
```

Beschreibung Algorithmus 1: Für die Bestimmung der Startknoten werden zunächst alle Knoten des Graphen G in die Menge der zu bestimmenden Startknoten *startNodes* als mögliche Lösung eingetragen. Anschließend wird über alle Kanten $e_i(f, s) \in \{P \cup R\}$ iteriert. Alle Knoten s der Kanten $e_i(f, s)$ werden aus der Menge der Startknoten entfernt. Somit sind alle Knoten, die einen Vorgänger infolge der Relation R oder V haben, keine Startknoten. Der Algorithmus terminiert nach vollständiger Iteration über

$\{P \cup R\}$ mit der Menge der vorhandenen Startknoten *startNodes*. Wird kein Startknoten identifiziert, liefert der Algorithmus dem Ergebnis entsprechend eine leere Menge zurück. Der Aufwand des angegebenen Algorithmus ist von der Anzahl n der Kanten in $\{P \cup R\}$ abhängig und ist von der Ordnung $O(n)$. Abbildung 3.3 zeigt das Ergebnis mit der gefundenen Knotenmenge $startNodes = \{1\}$ für den Beispielgraphen G.

Endknoten

Als Endknoten des Graphen G werden diejenigen Knoten bezeichnet, die keine wegführenden Kanten haben - oder anders ausgedrückt: Endknoten sind die Menge aller Knoten $b \in B$ oder $t \in T$, die keinen Nachfolger haben. Für alle Endknoten gilt:

$$endNodes := \{ek_i \in \{B \cup T\} \mid g_N(ek_i) = 0\} \tag{3.4}$$

Algorithmus 2 bestimmt nach Formel 3.4 die Menge der Endknoten. Als Eingangswert dient der auf das Vorhandensein von Endknoten zu untersuchende bipartite Graph G. Als Ergebnis wird die Menge der identifizierten Endknoten zurückgeliefert.

Algorithmus 2: getEndNodes(G)
Bestimmung der Menge von Endknoten

input : BipartiteGraph $G := (T, B; P, R)$
output : Menge der Endknoten *endNodes*

```
1 endNodes := {B ∪ T};
2 foreach Edge e_i(f, s) ∈ {P ∪ R} do
3 |   endNodes := endNodes \ f ;        // trage f aus der Menge der
  |   Endknoten aus
4 end
5 return endNodes
```

Beschreibung Algorithmus 2: Für die Bestimmung der Endknoten werden zunächst alle Knoten des Graphen G in die Menge der zu bestimmenden Endknoten *endNodes* als mögliche Lösung eingetragen. Anschließend wird über alle Kanten $e_i(f, s) \in \{P \cup R\}$ iteriert. Alle Knoten f der Kanten $e_i(f, s)$ werden aus der Menge der Endknoten entfernt. Somit sind alle Knoten, die einen Nachfolger infolge der Relation R oder P haben, keine Endknoten. Der Algorithmus terminiert nach vollständiger Iteration über $\{P \cup R\}$ mit der Menge der vorhandenen Endknoten *endNodes*. Wird kein Endknoten identifiziert, liefert der Algorithmus dem Ergebnis entsprechend eine leere Menge zurück. Der Aufwand des angegebenen Algorithmus ist von der Anzahl n der Kanten in $\{P \cup R\}$ abhängig und ist von der Ordnung $O(n)$.

Beispiel: Abbildung 3.3 zeigt das Ergebnis mit der gefundenen Knotenmenge *endNodes* $= \{6\}$ für den Beispielgraphen G.

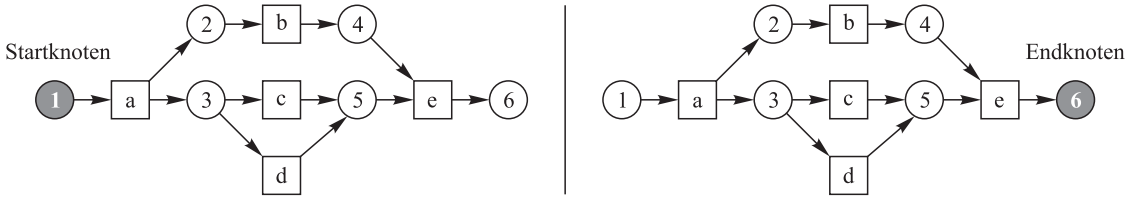


Abbildung 3.3: Beispiel für Start- und Endknoten in G

3.2.2 Wege ausgehend von einem Knoten oder zwischen zwei Knoten

Als Weg in einem Graphen wird ein offener Kantenzug[2] bezeichnet, der keinen Knoten mehrfach enthält.
Algorithmus 3 bestimmt als Ergebnis die Menge von Wegen M_w. Für den an dieser Stelle gezeigten Algorithmus ist ein Startknoten x_1 und optional ein Endknoten x_2 vorzugeben. Ist kein Endknoten x_2 als Eingangswert gegeben, ergibt sich der jeweilige Endknoten eines jeden Weges aus dem Knoten s der letzten Kante $e_i(f, s) \in \{R \cup P\}$ im gefundenen Weg. Ein Weg wird dabei als eine Folge[3] von Knoten abgebildet.

Beschreibung Algorithmus 3: Als Eingangswerte des Algorithmus dienen der zu betrachtende bipartite Graph G sowie ein Startknoten x_1, von dem ausgehend alle Wege im Graphen bestimmt werden sollen. Optional kann ein Endknoten x_2 angegeben werden. In diesem Fall wird ein gefundener Weg nur dann zur Menge der gefundenen Wege hinzugefügt, wenn über diesen der Endknoten x_2 auch erreicht wird.
Es werden zwei Stapelspeicher definiert. Der Stapelspeicher *pathStack* hält alle zu betrachtenden Wege, die noch nicht vollständig sind - in anderen Worten, deren Endknoten noch nicht bestimmt oder für die x_2 noch nicht erreicht wurde. Der Stapelspeicher *nodeStack* hält alle als nächstes zu betrachtende Knoten.
Es folgt die Initialisierung der beiden Stapelspeicher. Zunächst wird der gegebene Startknoten x_1 auf den Stapelspeicher *nodeStack* gelegt. Eine leere Knotenliste w_1 wird erzeugt und auf den Stapelspeicher *pathStack* gelegt.
Solange der Stapelspeicher *nodeStack* nicht leer ist, wird geprüft, ob der aktuell betrachtete Knoten *currentNode* Nachfolger hat. Ist dies der Fall und der aktuell betrachtete Knoten wurde noch nicht betreten, also nicht Element des momentan betrachteten Weges *currentPath* vom Stapelspeicher *pathStack*, wird er dem aktuellen Weg *currentPath* hinzugefügt. Der Weg wird im Anschluss daran wieder auf den Stapelspeicher *pathStack* gelegt. Der den Nachfolger repräsentierende Knoten wird entsprechend auf den Stapelspeicher *nodeStack* gelegt. Dies geschieht für alle Nachfolger *currentSuccessor* des betrachteten Knotens *currentNode*. Die Anzahl möglicher Wege erhöht sich dabei um die Anzahl der Nachfolger minus eins.
Hat der momentan betrachtete Knoten *currentNode* keinen Nachfolger und x_2 ist nicht gegeben, wurde ein vollständiger Weg ausgehend von x_1 bestimmt, falls der Weg aus

[2]Kantenzug, der mit verschiedenen Knoten beginnt und endet (siehe [Pahl u. Damrath 2000])
[3]sequentielle Auflistung der Knoten eines Weges

Algorithmus 3: getPathsBetween(G, x_1, x_2)
Wege ab einem oder zwischen zwei Knoten

input : BipartiteGraph $G := (T, B; P, R)$ und $x_1, x_2 \mid x_i \in \{B \cup T\}$
output : Wege *paths* ab x_1 oder zwischen x_1 und x_2

```
1  paths = ∅;
2  new Stack nodeStack;
3  new Stack pathStack;
4  knotenStack.push(x1);
5  wegeStack.push(neue Knotenliste w1 <>);
6  while knotenStack ≠ ∅ do
7      currentNode :=nodeStack.pop();
8      currentPath := pathStack.pop();
9      if (x2 = null ∧ gN(currentPath) = 0) ∨ (¬(x2 = null) ∧ x2 = currentNode)
       then
10         // trage currentNode in currentPath ein
11         currentPath := currentPath ∪ {currentNode};
12         // trage currentPath in paths ein
13         paths := paths ∪ {currentPath};
14         // Platzhalter - Siehe Kapitel 3.2.3
15     end
16     else
17         if currentNode ∉ currentPath then
18             currentPath := currentPath ∪ {currentNode};
19             successors := tN(currentNode);
20             foreach currentSuccessor ∈ successors do
21                 new NodeList w;
22                 // trage alle Knoten aus currentPath in w ein
23                 w := w ∪ currentPath;
24                 nodeStack.push(currentSuccessor);
25                 pathStack.push(w);
26             end
27         end
28     end
29 end
30 return paths
```

mehr Knoten als nur dem Startknoten besteht. Ist alternativ x_2 gegeben und *currentNode* entspricht dem Knoten x_2, wurde ebenfalls ein vollständiger Weg bestimmt. Nach dem Einfügen des momentan betrachteten Knotens *currentNode* in den momentan betrachteten Weg *currentPath* wird der Weg der Ergebnismenge *paths* hinzugefügt. Der Algorithmus terminiert, wenn keine Knoten mehr auf dem Stapelspeicher *knotenStack* liegen.
Wird kein Weg identifiziert, liefert der Algorithmus dem Ergebnis entsprechend eine leere Menge zurück.
Abbildung 3.4 zeigt das Ergebnis bezogen auf den Beispielgraphen mit den gefundenen Wegen $w_1 =< 1, a, 2, b, 4, e, 6 >$, $w_2 =< 1, a, 3, c, 5, e, 6 >$ und $w_3 =< 1, a, 3, d, 5, e, 6 >$ für den gegebenen Startknoten $x = 1$.

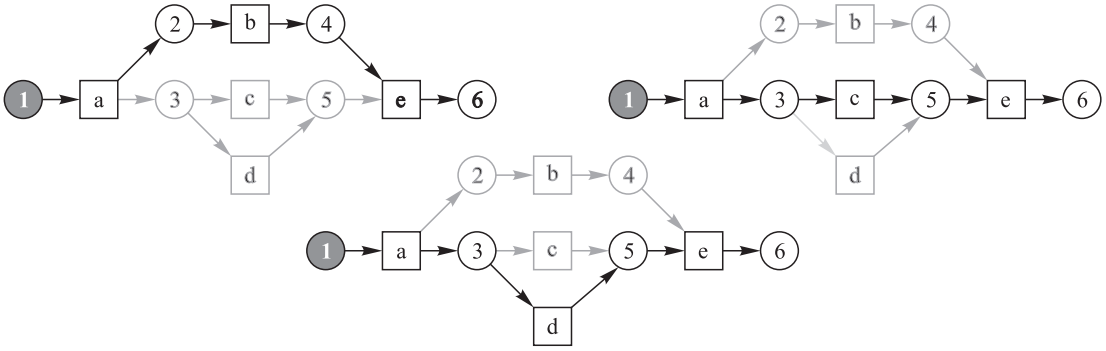


Abbildung 3.4: Beispiel für Wege ausgehend von einem Knoten $x_1 = 1$

Beispiel: Abbildung 3.5 zeigt das Ergebnis bezogen auf den Beispielgraphen mit den gefundenen Wegen $w_1 =< 1, a, 3, c, 5 >$ und $w_2 =< 1, a, 3, d, 5 >$ für den gegebenen Startknoten $x_1 = 1$ und den gegebenen Endknoten $x_2 = 5$.

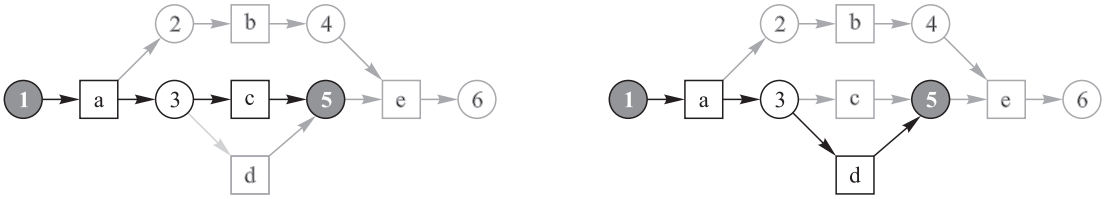


Abbildung 3.5: Beispiel für Wege zwischen zwei Knoten $x_1 = 1$ und $x_2 = 5$

3.2.3 Zufälliger Weg ab einem oder zwischen zwei Knoten

Unter einem zufälligen Weg wird die willkürliche Wahl eines Weges ab einem oder zwischen zwei Knoten verstanden. Dies ist beispielsweise für Algorithmus 5 in Kapitel 3.2.4 notwendig. Dieser verlangt das zufällige Wählen eines Weges zwischen zwei Knoten. Der an dieser Stelle vorgestellte Algorithmus 4 modifiziert zu diesem Zweck Algorithmus 3 aus Gründen der Effizienz.

Beschreibung Algorithmus 4: Für die Bestimmung eines zufälligen Weges wird Zeile 14 in Algorithmus 3 durch durch die in Algorithmus 4 angegebene ersetzt. Durch

Algorithmus 4: getRandomPath(G, x_1, x_2)
Zufälliger Weg ab x_1 oder zwischen x_1 und x_2

input : BipartiteGraph $G := (T, B; P, R)$ und $x_1, x_2 \mid x_i \in B \vee x_i \in T$
output : Wege *paths* ab x_1 oder zwischen x_1 und x_2

```
...;                                  // Fehlende Zeilen Siehe Algorithmus 3
14 return paths
...;                                  // Fehlende Zeilen Siehe Algorithmus 3
30 return paths
```

diese Modifikation wird der Algorithmus nach der Identifikation des ersten identifizierten Weges abgebrochen und der zu diesem Zeitpunkt bestimmte Weg als Element der Menge *paths* zurückgeliefert. Auch hier ist das Ergebnis eine leere Menge, falls kein solcher Weg existiert.

3.2.4 Kantendisjunkte Wege zwischen zwei Knoten

Als kantendisjunkte Wege zwischen einem Startknoten x_1 und einem Endknoten x_2 werden diejenigen Wege $w_i(x_1, x_2)$ bezeichnet, die keine gemeinsamen Kanten haben. Eine Menge von kantendisjunkten Wegen $W(x_1, x_2)$ wird dann als solche bezeichnet, wenn alle Wege dieser Menge paarweise kantendisjunkt sind. Dabei ist die Anzahl der möglichen kantendisjunkten Wege zwischen zwei Knoten x_1 und x_2 durch das Minimum vom Nachfolgergrad des Knotens x_1 und dem Vorgängergrad des Knotens x_2 $maxw(x_1, x_2) \leq Min\{g_N(x_1), g_V(x_2)\}$ beschränkt [Pahl u. Damrath 2000].
Algorithmus 5 basiert im Wesentlichen auf [Pahl u. Damrath 2000, Kapitel 8.4.4]. Der hauptsächliche Unterschied zur dort angegebenen Bildungsvorschrift besteht im Verfahren für die Bildung der Ersatzwege $\bar{w}_i$ und $\bar{w}_k$. Im Gegensatz zu [Pahl u. Damrath 2000] erfolgt hier die Prüfung nicht auf Basis einzelner, umgekehrter Kanten von w_i in w_k, sondern es werden umgekehrte Wege von w_i in w_k für die Bildung der Ersatzwege herangezogen. Es handelt sich hierbei um einen iterativen Vorgang, der für alle nicht zusammenhängenden Wege zwischen den alternierenden Ersatzwegen $\bar{w}_i$ und $\bar{w}_k$ ausgeführt werden muss.

Beschreibung Algorithmus 5: Es handelt sich bei Algorithmus 5 um einen rekursiven Algorithmus. Demzufolge benötigt er als Eingangswerte nicht nur den Graphen G und die beiden Knoten x_1 und x_2, sondern ebenfalls die Menge der zu diesem Zeitpunkt identifizierten, wahrscheinlichen kantendisjunkten Wege. Für den Start des iterativen Prozesses handelt es sich um eine leere Menge. Für die Konstruktion der kantendisjunkten Wege zwischen Knoten x_1 und Knoten x_2 wird zunächst ein beliebiger Weg zwischen den beiden Knoten bestimmt (siehe Algorithmus 4). Konnte kein Weg bestimmt werden, so gibt es auch keine weiteren oder gar keine kantendisjunkten Wege zwischen den Knoten. Für diesen Aufruf des Algorithmus wird der Menge der kantendisjunkten Wege *disjointPaths* kein weiterer hinzugefügt.
Ist die Menge der kantendisjunkten Wege *disjointPaths* keine leere Menge, muss der aktuell gefundene Weg w_i mit allen darin enthaltenen Wegen w_k verglichen werden.

Algorithmus 5: getEdgeDisjointPaths(G, x_1, x_2)
Kantendisjunkte Wege zwischen zwei Knoten

input : BipartiteGraph $G := (T, B; P, R)$ und $x_1, x_2 \mid x_i \in \{B \cup T\}$
output : *disJointPaths* ; `// kantendisjunkte Wege zwischen` x_1 `und` x_2

1 $w_i :=$ `getRandomPath(`G, x_1, x_2`)` ; `// Siehe Algorithmus 4`
2 **if** $|w_i| = 0$ **then return** *disjointPaths*;
3 $w_{original} := w_i$;
4 **if** $|disjointPaths| > 0$ **then**
5 **foreach** $w_k \in disjointPaths$ **do**
6 **while** *changed* = **true do**
7 *changed* := **false**;
8 `// Prüfe auf umgekehrte Kantenzüge`
9 **label** out: **for** $endW2 := |w_i| - 1$ **downto** 0 **do**
10 **for** $startW1 := 0$ *to* $|w_k|$ **do**
11 **if** $node_{endW2} \in w_i = node_{startW1} \in w_k$ **then**
12 `// gemeinsamen Knoten gefunden`
13 $endW1 := startW1$; $startW2 := endW2$;
14 **for** $j := endW2 - 1$ **bis** $j \geq 0 \wedge endW1 + 1 < |w_k|$ **step** -1 **do**
15 **if** $node_j \in w_i = node_{endW1+1} \in w_k$ **then**
16 `// Begin Kantenzug`
17 $startW2 := j$;
18 $endW1 := endW1 + 1$;
19 changed := **true**;
20 **end**
21 **else break** out ; `// verlasse Schleife label out`
22 **end**
23 **end**
24 **end**
25 **end**
26 **if** *changed* = **true** ; `// Ersetze` w_i `durch` $\bar{w_i}$ `und` w_k `durch` $\bar{w_k}$
27 **then**
28 $substW1 :=< node_i \mid node_i \in w_k \wedge 0 \leq i \leq startW1 + 1 >$;
29 $substW1 \circ < node_i \mid node_i \in w_i \wedge endW2 + 1 \leq i \leq |w_i| >$;
30 $substW2 :=< node_i \mid node_i \in w_i \wedge 0 \leq i \leq startW2 + 1 >$;
31 $substW2 \circ < node_i \mid node_i \in w_k \wedge endW1 + 1 \leq i \leq |w_k| >$;
32 $w_i := substW2$;
33 $pathsToRemove \cup \{w_k\}$;
34 $pathsToAdd \cup \{substW1\}$;
35 **end**
36 **end**
37 $disjointPaths \cup pathsToAdd$;
38 $disjointPaths \setminus pathsToRemove$;
39 **end**
40 **end**
41 Bilde Ersatzgraph G_i durch Umkehrung der Kanten von $w_{original}$ in G;
42 $disjointPaths \cup \{w_i\}$;
43 `getEdgeDisjointPaths(`$G_i, x_1, x_2, disjointPaths$`)` ; `// Rekursion`
44 **return** *disJointPaths*;

Es wird auf das erste Auftreten eines umgekehrten Kantenzuges von w_i in $w_k \in$ *disjointPaths* geprüft. Es wird w_i durch $\bar{w}_i$ und w_k durch $\bar{w}_k$ ersetzt. Dieser Schritt wird solange mit den jeweiligen Ersatzwegen wiederholt, bis es keine umgekehrten Kantenzüge von $\bar{w}_i$ in $\bar{w}_k$ mehr gibt.

Beispiel:

$$w_i =< x_1, \ldots, a, l, i, v, e, \ldots, u, t, r, f, \ldots, x_2 >$$
$$w_k =< x_1, \ldots, s, v, i, l, m, \ldots, b, r, t, m, \ldots, x_2 >$$

Erster Iterationsschritt

Die Kante (r, t) von w_k ist in umgekehrter Richtung in w_i enthalten:

$$w_i =< x_1, \ldots, a, l, i, v, e, \ldots, u, \mathbf{t,r}, f, \ldots, x_2 >$$
$$w_k =< x_1, \ldots, s, v, i, l, m, \ldots, b, \mathbf{r,t}, m, \ldots, x_2 >$$

Es folgt die Bildung der verkürzten Wege $\bar{w}_i$ über die Kettung des ersten Teilweges $< x_1, \ldots, a, l, i, v, e, \ldots, u, t >$ von w_i und dem zweiten Teilweg $< m, \ldots, x_2 >$ von w_k und $\bar{w}_k$ über die Kettung des ersten Teilweges $< x_1, \ldots, s, v, i, l, m, \ldots, b, r >$ von w_k mit dem zweiten Teilweg $< f, \ldots, x_2 >$ von w_i:

$$\bar{w}_i =< x_1, \ldots, a, l, i, v, e, \ldots, u, t, m, \ldots, x_2 >$$
$$\bar{w}_k =< x_1, \ldots, s, v, i, l, m, \ldots, b, r, f, \ldots, x_2 >$$

Zweiter Iterationsschritt - für $\bar{w}_i$ und $\bar{w}_k$

Der Weg $< v, i, l >$ von $\bar{w}_k$ ist in umgekehrter Richtung in $\bar{w}_i$ enthalten:

$$\bar{w}_i =< x_1, \ldots, a, \mathbf{l,i,v}, e, \ldots, u, t, r, t, m, \ldots, x_2 >$$
$$\bar{w}_k =< x_1, \ldots, s, \mathbf{v,i,l}, m, \ldots, b, r, f, \ldots, x_2 >$$

Es folgt die Bildung der verkürzten Wege $\bar{w}_i$ über die Kettung des ersten Teilweges $< x_1, \ldots, a, l >$ von $\bar{w}_i$ mit dem zweiten Teilweg $< m, \ldots, b, r, f, \ldots, x_2 >$ von $\bar{w}_k$ aus dem vorherigen Iterationsschritt und $\bar{w}_k$ über die Kettung des ersten Teilweges $< x_1, \ldots, s, v >$ von $\bar{w}_k$ mit dem zweiten Teilweg $< e, \ldots, u, t, m, \ldots, x_2 >$ von $\bar{w}_i$ aus dem vorherigen Iterationsschritt:

$$\bar{w}_i =< x_1, \ldots, a, l, m, \ldots, b, r, f, \ldots, x_2 >$$
$$\bar{w}_k =< x_1, \ldots, s, v, e, \ldots, u, t, r, t, m, \ldots, x_2 >$$

Dritter Iterationsschritt - für $\bar{w}_i$ und $\bar{w}_k$

Es sind keine weiteren umgekehrten Wege von $\bar{w}_k$ in $\bar{w}_i$ enthalten, somit endet die Iteration.

Im Anschluss wird der modifizierte Weg $\bar{w}_k$ in der Ergebnismenge *disjointPaths* ausgetauscht und der modifizierte Weg $\bar{w}_i$ eingetragen.
Es folgt der rekursive Aufruf des Algorithmus mit dem Ersatzgraphen G_i. Dieser wird gebildet, indem alle im Weg $w_{original}$ (entspricht dem ursprünglich gefunden Weg w_i) enthaltenen Kanten im Graph G umgekehrt werden. Der Algorithmus terminiert, wenn kein Weg zwischen x_1 und x_2 im jeweiligen Ersatzgraph G_i mehr gefunden wird. Die Menge *disjointPaths* entspricht dann der Menge kantendisjunkter Wege zwischen x_1 und x_2.
Abbildung 3.6 zeigt als Beispiel die kantendisjunkten Wege $w_1 =< a, 2, b, 3, c >$ und $w_2 =< a, 1, b, 4, c >$ zwischen Knoten $x_1 = a$ und Knoten $x_2 = c$ für den dargestellten Graphen G.

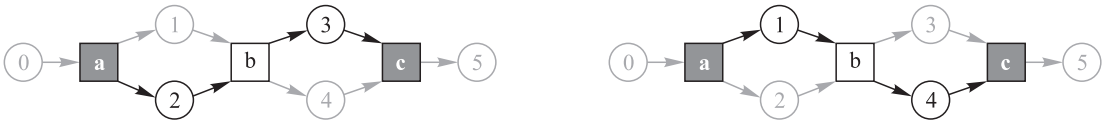


Abbildung 3.6: Beispiel für kantendisjunkte Wege zwischen zwei Knoten

3.2.5 Knotendisjunkte Wege zwischen zwei Knoten

Als knotendisjunkte Wege innerhalb eines Graphen zwischen einem Startknoten x_1 und einem Endknoten x_2 werden diejenigen Wege bezeichnet, die außer einem gemeinsamen Start- und Endknoten keinen weiteren gemeinsamen Knoten haben. Eine Menge von knotendisjunkten Wegen zwischen einem Knoten x_1 und einem Knoten x_2 wird als solche bezeichnet, wenn die enthaltenen Wege untereinander paarweise disjunkt sind. Dabei ist die Anzahl der möglichen knotendisjunkten Wege zwischen den Knoten x_1 und x_2 durch das Minimum vom Nachfolgergrad des Knotens x_1 und dem Vorgängergrad des Knotens x_2 $maxw(x_1, x_2) \leq Min\{g_N(x_1), g_V(x_2)\}$ beschränkt [Pahl u. Damrath 2000].
Algorithmus 6 basiert auf [Pahl u. Damrath 2000, Kapitel 8.4.4] und dient der Bestimmung der Menge knotendisjunkter Wege zwischen einem Startknoten x_1 und einem Endknoten x_2.

Beschreibung Algorithmus 6: Als Eingangswerte des Algorithmus dienen der zu betrachtende bipartite Graph G sowie ein Startknoten x_1 und ein Endknoten x_2, zwischen denen alle knotendisjunkte Wege im Graphen bestimmt werden sollen. Es wird der Ersatzgraph G_E gebildet, indem zunächst jeder Knoten $node_i \in G$ durch eine Kante $(node'_i, node''_i)$ ersetzt wird. Diese Substitution wird in der Relation *substitutionMap* durch Abbildung von $node_i \in \{B \cup T\}$ auf die Ersatzkante $(node'_i, node''_i)$ des Knotens in G_E gespeichert, um das Einfügen der Kanten des Graphen G zu ermöglichen. Weiterhin wird die umgekehrte Abbildung *reverseSubstitutionMap* der Ersatzkanten auf den ursprünglichen Knoten gespeichert, um die Substitution der Ersatzkanten auf die Ursprungsknoten zu ermöglichen.
Über den Aufruf des Algorithmus 5 werden alle kantendisjunkten Wege *edgeDisjointPaths* im Ersatzgraphen G_E bestimmt. Durch die Rücksubstitution der Ersatzkanten eines jeden Weges $currentPath \in edgeDisjointPaths$ werden die knotendisjunkten Wege

Algorithmus 6: getNodeDisjointPaths(G, x_1, x_2)
Knotendisjunkte Wege zwischen Knoten

input : BipartiteGraph $G := (T, B; P, R)$ und $x_1, x_2 \mid x_i \in \{B \cup T\}$
output : knotendisjunkte Wege *nodeDisjointPaths* zwischen x_1 und x_2

```
1  // Bilde Ersatzgraphen G_E
2  G_E := (T_E, B_E; P_E, R_E);
3  // Relation der Knoten von G auf die Kanten im Ersatzgraphen
       G_E
4  substitutionMap := N × P_E | N ∈ {B ∪ T};
5  // Relation der Kanten im Ersatzgraphen G_E auf die Knoten von
       G
6  reverseSubstitutionMap := P_E × N | N ∈ {B ∪ T};
7  // Ersetzen der Knoten von G durch Kanten in G_E
8  foreach node_i ∈ {B ∪ T} do
9      B_E := B_E ∪ {node'_i};
10     T_E := T_E ∪ {node''_i};
11     // Ersatzkante in P_E eintragen
12     P_E := P_E ∪ {(node'_i, node''_i)};
13     // Knoten in G auf Ersatzkante abbilden
14     substitutionMap := substitutionMap ∪ {(node_i, (node'_i, node''_i))};
15     // Ersatzkante in G_E auf Knoten in G abbilden
16     reverseSubstitutionMap :=
       reverseSubstitutionMap ∪ {((node'_i, node''_i), node_i)};
17 end
18 // Einfügen der Kanten im Ersatzgraphen
19 foreach edge_i(f_j, s_k) ∈ {P ∪ R} do
20     first_i = node''_n | (n_l, (node'_m, node''_n) ∈ substitutionMap ∧ n_l = f_j;
21     second_i = node'_m | (n_l, (node'_m, node''_n) ∈ substitutionMap ∧ n_l = s_k;
22     if first_i ∈ B_E then insert (first_i, second_i) in P_E
23     else trage (first_i, second_i) in R_E ein
24 end
25 edgeDisjointPaths := getEdgeDisjointPaths(G_i, x_1, x_2, disjointPaths);
26 nodeDisjointPaths := ∅;
27 foreach currentPath ∈ edgeDisjointPaths do
28     substitutionPath := ∅;
29     substitutionPath := substitutionPath ∪ {x_1};
30     for i := 1 to |currentPath| − 1 step +2 do
31         substNode_i := n_l | ((node'_m, node''_n), n_l) ∈ reverseSubstitutionMap ∧
           node'_m = node_i ∧ node''_m = node_{i+1} ∧ node_i ∈ currentPath ;
32     end
33     substitutionPath := substitutionPath ∪ {x_2};
34     nodeDisjointPaths := nodeDisjointPaths ∪ {substitutionPath};
35 end
36 return nodeDisjointPaths
```

in G bestimmt. Werden keine knotendisjunkte Wege identifiziert, liefert der Algorithmus dem Ergebnis entsprechend eine leere Menge zurück.

Beispiel: Abbildung 3.7 zeigt das Ergebnis bezogen auf den Beispielgraphen mit den gefundenen Wegen $w_1 = \{a, 3, c, 5, e\}$, $w_2 = \{a, 2, b, 4, e\}$ für den gegebenen Startknoten $x_1 = a$ und den gegebenen Endknoten $x_2 = e$ im oberen Teil der Abbildung. Im unteren Teil wird der bereits für die kantendisjunkten Wege verwendete Graph aus Abbildung 3.6 dargestellt. Hier gibt es nur einen Weg.

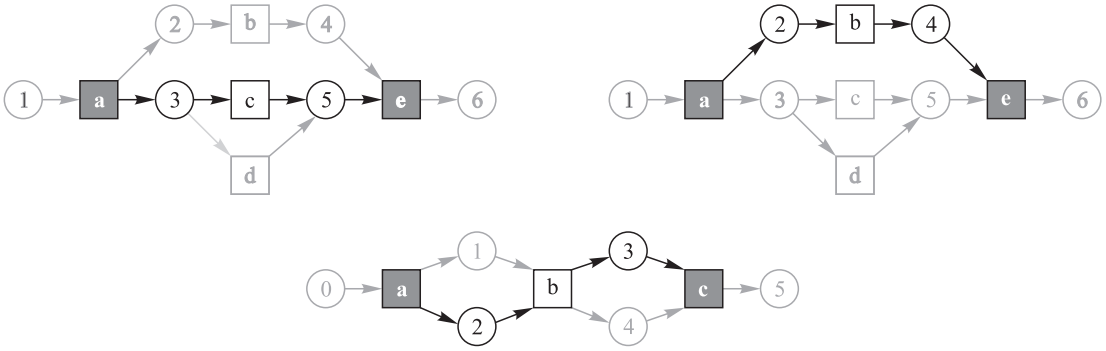


Abbildung 3.7: Beispiel für knotendisjunkte Wege zwischen zwei Knoten

3.2.6 Subgraphen ab einem Knoten oder zwischen zwei Knoten

Als Subgraph G_{sub} wird im Rahmen dieser Arbeit ein Graph G_{sub} bezeichnet, der alle Knoten und Kanten des Graphen G enthält, die ausgehend von einem gegebenen Knoten $x_1 \in \{B \cup T\}$ erreicht werden können. Dieser Knoten definiert demnach den Startknoten eines solchen Subgraphen. Wird als weitere Randbedingung der Endknoten $x_2 \in \{B \cup T\}$ des zu bestimmenden Subgraphen vorgegeben, erhält man die Menge aller Knoten und Kanten, über die ausgehend vom Startknoten der Endknoten erreicht werden kann.

Subgraphen ausgehend von einem Knoten

Ein Subgraph G_{sub} ausgehend von einem Knoten x_1 wird gebildet, indem zunächst alle Knoten der ermittelten Wege in G nach Algorithmus 3 mit $x_2 = null$ der jeweiligen Knotenmenge T_{sub} oder B_{sub} zugeordnet werden. Gleiches gilt für die entsprechenden Kanten $p_i(b, t) \in P$ und $r_i(t, b) \in R$, die jeweils zwischen den Knoten eines Weges existieren(siehe Kapitel 3.2.2).

Subgraphen zwischen zwei Knoten

Ein Subgraph G_{sub} zwischen einem Knoten $x_1 \in \{B \cup T\}$ und einem Knoten $x_2 \in \{B \cup T\}$ wird gebildet, indem zunächst alle Knoten der ermittelten Wege in G nach Algorithmus 3 mit x_1 und x_2 der jeweiligen Knotenmenge T_{sub} oder B_{sub} zugeordnet werden. Gleiches gilt für die entsprechenden Kanten $p_i(b, t) \in P$ und $r_i(t, b) \in R$, die jeweils zwischen den Knoten eines Weges existieren (siehe Kapitel 3.2.2).

Algorithmus 7 stellt die notwendigen Schritte für die Bestimmung von Subgraphen dar.

Algorithmus 7: getSubGraphBetween(G, x_1, x_2)
Subgraph zwischen oder ab Knoten

input : BipartiteGraph $G := (T, B; P, R)$ und x_1, x_2 $\quad | \; x \in \{B \cup T\}$
output : G_{sub} ab Knoten x_1 mit x_2=**null** oder zwischen Knoten x_1 und x_2

```
1  G_sub := (T_sub, B_sub; P_sub, R_sub);
2  paths := getPathsBetween(G, x_1, x_2);
3  foreach currentPath < node_i, node_{i+1}, ..., node_{|currentPath|} >∈ paths do
4      for i = 0 to i < |currentPath| step +1 do
5          currentNode := node_i;
6          if currentNode ∈ B then
7              B_sub := B_sub ∪ {currentNode};
8              if i + 1 < |currentPath| then
9                  P_sub ∪ {(currentNode, node_{i+1})};
10             end
11         end
12         else
13             T_sub := T_sub ∪ {currentNode};
14             if i+1 < |currentPath| then
15                 R_sub := R_sub ∪ {(currentNode, node_{i+1})};
16             end
17         end
18     end
19 end
20 return G_sub
```

Beschreibung Algorithmus 7: Als Eingangswerte des Algorithmus dienen der zu betrachtende bipartite Graph G sowie der Startknoten x_1 und ein optionaler Endknoten x_2, von dem aus bzw. zwischen denen der Subgraph bestimmt werden soll. Ist x_2 gegeben, werden alle Wege zwischen den Knoten x_1 und x_2 nach Algorithmus 3 bestimmt und in *paths* eingetragen. Ist x_2 nicht gegeben, erfolgt die Ermittlung der Wege ausgehend von Knoten x_1. Anschließend wird jeder Knoten *currentNode* eines jeden Weges in *paths* in die entsprechende Knotenmenge B_{sub} oder T_{sub} des Graphen G_{sub} eingetragen. Gleichzeitig werden die entsprechenden Kanten $p(b, t) \in P$ und $r(t, b) \in R$ in P_{sub} bzw. R_{sub} eingetragen. Handelt es sich bei *currentNode* um den Endknoten eines Weges, dann existiert keine Kante, die dem Graphen G_{sub} hinzugefügt werden muss.

Beispiel: Abbildung 3.8 zeigt das Ergebnis bezogen auf den Beispielgraphen mit den Subgraphen G_{sub_1}, für einen gegebenen Startknoten $x_1 = a$ (linke Seite) und G_{sub_2}, für den gegebenen Startknoten $x_1 = 3$ und den gegebenen Endknoten $x_2 = e$ (rechte Seite).

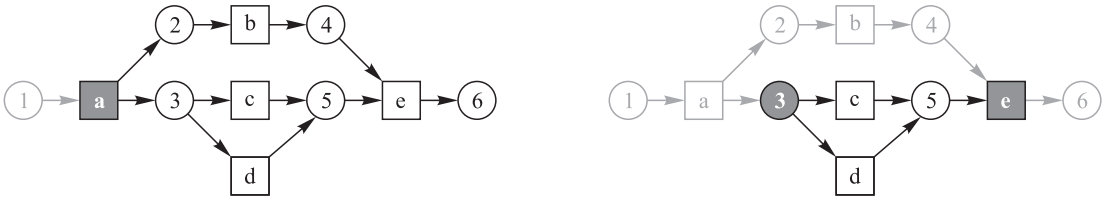


Abbildung 3.8: Beispiel für Subgraphen

3.3 Workflowgraphen

Im Rahmen dieser Arbeit wird als Workflowgraph formal ein bipartiter Graph nach Kapitel 3.1 verstanden. Er besteht aus einer Menge von Aktivitäten und einer Menge von Transitionen. Eine Transition ist der Übergang von vorangegangenen Aktivitäten zu nachfolgenden Aktivitäten. Eine Aktivität kann nur dann begonnen werden, wenn dessen Vorgängertransitionen erreicht wurden. Die nachfolgenden Definitionen basieren auf [van der Aalst 1998], [Klinger 2003], [König 2004].

Entscheidung/Asynchronisation Ein Workflowgraph kann Entscheidungen beinhalten (*XOR-split*), die den Ablauf im Graphen verzweigen oder im Gegensatz dazu Asynchronisationen (*AND-Split*), die das parallele Ausführen von Aktivitäten ermöglichen. Eine Entscheidung tritt auf, wenn eine Transition mehr als einen Nachfolger hat. Eine Asynchronisation ist durch eine Aktivität mit mehr als einem Nachfolger gekennzeichnet. Bei einer Entscheidung wird also genau eine der nachfolgenden Kanten betreten. Bei einer Asynchronisation werden hingegen alle nachfolgenden Kanten betreten.

Kontakt/Synchronisation Korrespondierend zu Entscheidungen und Asynchronisationen kann ein Workflowgraph Kontakt- (*XOR-Join*) und Synchronisationsknoten (*AND-Join*) enthalten. Ein Kontakt führt vorangestellte Entscheidungen wieder zusammen. Es handelt sich um Knoten der Menge der Transitionen, die mehr als einen Vorgänger haben. Für einen Kontakt wird demnach gefordert, dass mindestens eine seiner Vorgängeraktivitäten erreichbar ist. Eine Synchronisation ist eine Aktivität, die mehr als einen Vorgänger hat. Sie führt vorangestellte Asynchronisationen wieder zusammen. Es wird gefordert, dass alle Vorgängertransitionen erreichbar sind.

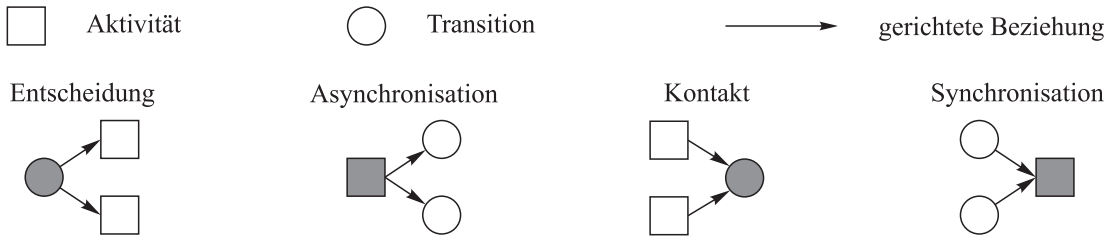


Abbildung 3.9: Workflowgraphen - Übersicht

3.4 Algorithmen und Methoden auf Workflowgraphen

Für die Darstellung der Methoden und Algorithmen auf Workflowgraphen wird weiterhin Graph G aus Kapitel 3.1 verwendet. Bezogen auf die Definitionen in Kapitel 3.3 handelt es sich bei Knotenmenge T um die Menge der Aktivitäten und Menge B stellt die Menge der Transitionen dar. Die Beziehung zwischen Aktivitäten und Transitionen wird durch die Kantenmenge P und die Beziehung zwischen Transitionen und Aktivitäten durch die Menge der Kanten R abgebildet.

3.4.1 Identifikation von XOR-Split- und XOR-Join-Knoten

XOR-Split- und XOR-Join-Knoten eines Workflowgraphen können durch Betrachtung der Struktur des Graphen bestimmt werden.

XOR-Split-Knoten

Als XOR-Split Knoten werden jene Knoten der Menge der Transitionen B bezeichnet, die nachfolgende Bedingung erfüllen:

Alle Knoten $b_i \in B$, die mehr als eine wegführende Kante besitzen, sind XOR-Split-Knoten. Oder anders ausgedrückt, alle Knoten $b \in B$, die mehr als einen Nachfolger besitzen, sind XOR-Split-Knoten:

$$xorSplits := \{b_i \in B \mid g_N(b_i) > 1\} \tag{3.5}$$

Algorithmus 8 bestimmt die Menge der XOR-Splits. Als Eingangswert dient der zu untersuchende bipartite Graph. Als Ergebnis wird die Menge der identifizierten XOR-Splits zurückgeliefert. Werden keine XOR-Splits identifiziert, liefert der Algorithmus eine leere Menge als Ergebnis.

Algorithmus 8: getXorSplits(G)
Bestimmung der Menge von XOR-Split-Knoten

input : BipartiteGraph $G := (T, B; P, R)$
output : Menge der XOR-Split Knoten *xorSplits*

```
1 xorSplits := ∅;
2 foreach possibleSplit ∈ B do
3 |   // hat possibleSplit mehr als einen Nachfolger ?
4 |   if g_N(possibleSplit) > 1 then
5 |   |   // trage possibleSplit in die Menge der XOR-Splits ein
6 |   |   xorSplits := xorSplits ∪ {possibleSplit};
7 |   end
8 end
9 return xorSplits
```

Beschreibung Algorithmus 8: Für die Bestimmung der XOR-Splits wird über alle $b_i \in B$ iteriert. Hat ein Knoten b_i mehr als einen Nachfolger wird er der Menge *xorSplits* hinzugefügt. Abbildung 3.10 zeigt die nach dem Algorithmus bestimmte Knotenmenge $xorSplits = \{3\}$.

XOR-Join-Knoten

Die Menge aller Knoten b_i der Menge der Transitionen B werden als XOR-Join-Knoten bezeichnet, wenn für sie mehr als eine hinführende Kante existiert, der Knoten b_i demnach mehr als einen Vorgänger hat. Für alle XOR-Joins gilt:

$$xorJoins := \{b_i \in B \mid g_V(b_i) > 1\} \tag{3.6}$$

Algorithmus 9 bestimmt die Menge der XOR-Joins. Als Eingangswert dient der auf das Vorhandensein von XOR-Joins zu untersuchende bipartite Graph. Als Ergebnis wird die Menge der identifizierten XOR-Joins zurückgeliefert.

Algorithmus 9: getXorJoins(G)
Bestimmung der Menge von XOR-Join-Knoten

input : BipartiteGraph $G := (T, B; P, R)$
output : Menge der XOR-Join Knoten *xorJoins*

```
1 xorJoins := ∅;
2 foreach possibleJoin ∈ B do
3 |   // hat possibleJoin mehr als einen Vorgänger ?
4 |   if |g_V(possibleJoin)| > 1 then
5 |   |   // trage possibleJoin in die Menge der XOR-Joins ein
6 |   |   xorJoins := xorJoins ∪ {possibleJoin};
7 |   end
8 end
9 return xorJoins
```

Beschreibung Algorithmus 9: Für die Bestimmung der XOR-Joins wird über alle $b_i \in B$ iteriert. Hat ein Knoten b_i mehr als einen Vorgänger, wird er der Menge *xorJoins* hinzugefügt.

Beispiel: Abbildung 3.10 zeigt die nach dem Algorithmus bestimmte Knotenmenge $xorJoins = \{5\}$.

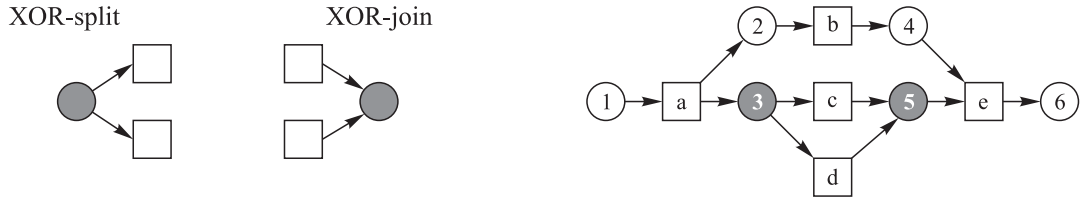


Abbildung 3.10: Beispiel für XOR-Split und XOR-Join Knoten

3.4.2 Identifikation von AND-Split und AND-Join Knoten

AND-Split- und AND-Join-Knoten eines Workflowgraphen können durch Betrachtung der Struktur des Graphen bestimmt werden.

AND-Split Knoten

Als AND-Split-Knoten werden jene Knoten der Menge der Aktivitäten T bezeichnet, die nachfolgende Bedingung erfüllen:

Alle Knoten t_i der Menge T, die mehr als eine wegführende Kante haben, sind AND-Split-Knoten, bzw. alle Knoten t_i der Menge T, die mehr als einen Nachfolger besitzen, sind AND-Split-Knoten:

$$andSplits := \{t_i \in T \mid g_N(t_i) > 1\} \tag{3.7}$$

Algorithmus 10 bestimmt die Menge der AND-Splits. Als Eingangswert dient der zu untersuchende bipartite Graph. Als Ergebnis wird die Menge der identifizierten AND-Splits zurückgeliefert. Werden keine AND-Splits identifiziert, liefert der Algorithmus eine leere Menge als Ergebnis.

Algorithmus 10: getAndSplits(G)
Bestimmung der Menge von AND-Split-Knoten

input : BipartiteGraph $G := (T, B; P, R)$
output : Menge der AND-Split Knoten *andSplits*

```
1 andSplits := ∅;
2 foreach possibleSplit ∈ T do
3 |   // hat possibleSplit mehr als einen Nachfolger ?
4 |   if g_N(t_i) > 1 then
5 |   |   // trage possibleSplit in die Menge der AND-Splits ein
  |   |       andSplits := andSplits ∪ t_i}
6 |   end
7 end
8 return andSplits
```

Beschreibung Algorithmus 10: Für die Bestimmung der AND-Splits wird über alle $t_i \in T$ iteriert. Hat ein Knoten t_i mehr als einen Nachfolger, wird er der Menge *andSplits* hinzugefügt. Abbildung 3.11 zeigt die nach dem Algorithmus bestimmte Knotenmenge $andSplits = \{a\}$.

AND-Join Knoten

Alle Knoten p_i der Menge der Aktivitäten T werden als AND-Join Knoten bezeichnet, wenn für sie mehr als eine hinführende Kante existiert - der Knoten t_i demnach mehr

als einen Vorgänger hat. Für alle AND-Joins gilt:

$$andJoins := \{t_i \in T \mid g_V(t_i) > 1\} \tag{3.8}$$

Algorithmus 11 bestimmt die Menge der AND-Joins. Als Eingangswert dient der zu untersuchende bipartite Graph. Als Ergebnis wird die Menge der identifizierten AND-Joins zurückgeliefert.

Algorithmus 11: getAndJoins(G)
Bestimmung der Menge von AND-Join-Knoten

input : BipartiteGraph $G := (T, B; P, R)$
output : **return** Menge der AND-Join Knoten *andJoins*

```
1 andJoins = ∅;
2 foreach possibleJoin ∈ T do
3 |   // hat possibleJoin mehr als einen Vorgänger ?
4 |   if |g_V(possibleJoin)| > 1 then
5 |   |   // trage possibleJoin in die Menge der AND-Splits ein
6 |   |   andJoins := andJoins ∪ {t_i};
7 |   end
8 end
9 return andJoins
```

Beschreibung Algorithmus 11: Für die Bestimmung der AND-Joins wird über alle $t_i \in T$ iteriert. Hat ein Knoten t_i mehr als einen Vorgänger, wird er der Menge *andJoins* hinzugefügt.

Beispiel: Abbildung 3.11 zeigt die nach dem Algorithmus bestimmte Knotenmenge $andJoins = \{e\}$.

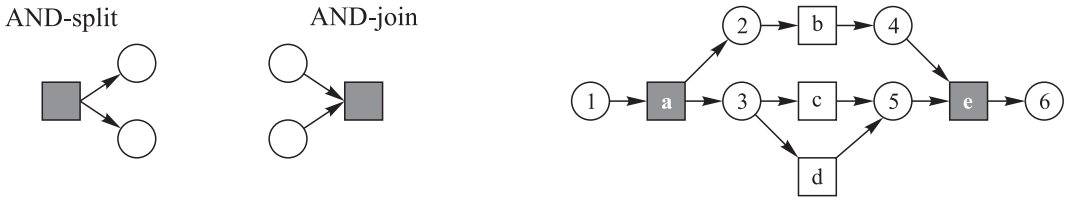


Abbildung 3.11: Beispiel für AND-Split und AND-Join Knoten

3.4.3 Identifikation korrespondierender Split-Join und Join-Split Paare

In einem Workflowgraphen, der *well handled* ist (siehe Kapitel 3.4.4), hat jeder XOR-Split und jeder AND-Split einen korrespondierenden Join-Knoten, durch den er synchronisiert wird. Welcher split-Knoten *split* zu welchem Join-Knoten *join* und umgekehrt gehört, kann durch Algorithmus 12 bzw. 13 bestimmt werden.

Algorithmus 12: getJoinsFromSplit(G, *split*)
Bestimmung der Menge korrespondierender Join-Knoten von einem Split-Knoten

input : BipartiteGraph $G := (T, B; P, R)$, *split*
output : Menge der korrespondierenden join-Knoten *correspondingJoins*

```
1  correspondingJoins := ∅;
2  possibleJoins := ∅;
3  if split ∈ getANDSplits(G) then
4  |  possibleJoins := getANDJoins(G)
5  end
6  else possibleJoins := getXORJoins(G);
7  foreach currentJoin ∈ possibleJoins do
8  |  if |getNodeDisjointPaths (G, split, currentJoin)| > 1 then
9  |  |  correspondingJoins := correspondingJoins ∪ {currentJoin}
10 |  end
11 end
12 return correspondingJoins
```

Bestimmung von korrespondierenden Join-Knoten

Die Menge von Join-Knoten, die einen Split-Knoten synchronisieren, kann durch Algorithmus 12 bestimmt werden. Dabei wird die Art der Synchronisation beachtet. AND-Join Knoten werden für einen AND-Split Knoten bestimmt und XOR-Join Knoten für einen XOR-Split Knoten.

Beschreibung Algorithmus 12: Für die Bestimmung der Menge der korrespondierenden Synchronisationen *correspondingJoins* nach Algorithmus 12 wird zunächst festgestellt, ob es sich beim Knoten *split* um einen AND-Split oder einen XOR-Split handelt. Da davon ausgegangen wird, dass es sich bei G um einen Graphen handelt, der *well handled* ist, sind nur passende Join-Knoten möglich - also AND-Split mit AND-Join und XOR-Split mit XOR-Join. Die entsprechende Menge von Knoten *possibleJoins* wird nach Algorithmus 11 oder 9 bestimmt. Anschließend wird paarweise geprüft, ob mehr als ein knotendisjunkter Weg zwischen Knoten *split* und jedem Knoten *currentJoin* der Menge *possibleJoins* existiert. Trifft diese Bedingung für einen Knoten *currentJoin* zu, wird dieser in die Ergebnismenge *correspondingJoins* eingetragen. Abbildung 3.12 zeigt die Ergebnisse *correspondingJoins* $= \{5\}$ und *correspondingJoins* $= \{e\}$ des Algorithmus für *split* $= 3$ bzw. *split* $= a$.

Bestimmung von korrespondierenden Split-Knoten

Die Menge von Split-Knoten, die durch einen Join-Knoten synchronisiert werden, kann durch Algorithmus 13 bestimmt werden. Dabei wird die Art der Synchronisation beachtet. AND-Split Knoten werden für einen AND-Join Knoten bestimmt und XOR-Split Knoten für einen XOR-Join Knoten.

Algorithmus 13: getSplitsFromJoin($G, join$)
Bestimmung der Menge von korrespondierenden split-Knoten von einem join-Knoten

input : BipartiteGraph $G := (T, B; P, R)$, *join*
output : der korrespondierenden Join-Knoten *correspondingJoins*

```
1  correspondingSplits := ∅;
2  possibleSplits := ∅;
3  if join ∈ getANDJoins(G) then
4  |  possibleSplits := getANDSplits(G)
5  end
6  else possibleSplits := getXORSplits(G);
7  foreach currentSplit ∈ possibleSplits do
8  |  if |getNodeDisjointPaths (G, currentSplit, join)| > 1 then
9  |  |  correspondingSplits := correspondingSplits ∪ {currentSplit}
10 |  end
11 end
12 return correspondingSplits
```

Beschreibung Algorithmus 13: Für die Bestimmung der Menge der korrespondierenden Splits *correspondingSplits* wird zunächst festgestellt, ob es sich beim Knoten *join* um einen AND-Join oder einen XOR-Join handelt. Da davon ausgegangen wird, dass es sich bei G um einen Graphen handelt, der *well handled* ist, sind nur passende Split-Knoten möglich - also AND-Join mit AND-Split und XOR-Join mit XOR-Split. Die entsprechende Menge von Knoten *possibleSplits* wird nach Algorithmus 10 oder 8 bestimmt. Anschließend wird paarweise geprüft, ob mehr als ein knotendisjunkter Weg zwischen jedem Konten der Menge *currentSplit* und dem Knoten *join* existiert. Trifft diese Bedingung für einen Knoten *currentSplit* zu, wird dieser in die Ergebnismenge *correspondingSplits* eingetragen.

Beispiel: Abbildung 3.12 zeigt die Ergebnisse *correspondingSplits* $= \{3\}$ und *correspondingSplits* $= \{a\}$ des Algorithmus für *join* $= 5$ bzw. *join* $= e$.

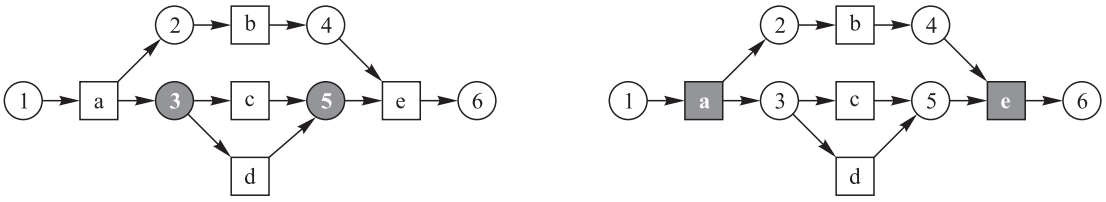


Abbildung 3.12: Beispiel für korrespondierende split- / join-Paare

3.4.4 Well Handled Test und Bestimmung von Konfliktpaaren

Ein Workflowgraph ist dann *well handled*, wenn jeder AND-Split mit einem AND-Join und jeder XOR-Split mit einem XOR-Join synchronisiert wird (siehe [Richter von

Hagen u. Stucky 2004]). Ist dies nicht der Fall, kann dies zu Verklemmungen im Ablauf des Graphen führen. Ist ein Workflowgraph *well handled*, so ist seine Struktur korrekt und er enthält nur gültige Instanzgraphen und somit ausschließlich gültige Abläufe. Ein Workflowgraph ist nicht *well handled*, wenn knotendisjunkte Wege zwischen artfremden Split-Join-Paaren im Graphen existieren. Ein solches Paar wird Konfliktpaar genannt.

Beschreibung Algorithmus 14: Für die Bestimmung der Konfliktpaare werden zunächst alle AND-Joins *andJoins* nach Algorithmus 11 und alle XOR-Splits *xorSplits* nach Algorithmus 8 bestimmt. Anschließend erfolgt die Ermittlung der Menge knotendisjunkter Wege paarweise zwischen jedem Knoten $currentXorSplit \in xorSplits$ und jedem Knoten $currentAndJoin \in andJoins$. Ist der Betrag der Menge knotendisjunkter Wege - also deren Anzahl - größer eins, wurde ein Konfliktpaar (*currentXorSplit*, *currentAndJoin*) identifiziert und wird der Ergebnismenge *conflictPairs* hinzugefügt. Dieser Prozess wird für alle nach Algorithmus 10 bestimmten AND-Split Knoten mit jedem nach Algorithmus 9 bestimmten XOR-Join Knoten wiederholt. Als Ergebnis wird die Menge von Konfliktpaaren *conflictPairs* zurückgeliefert. Handelt es sich dabei um eine leere Menge, ist der Graph *well handled*, anderenfalls nicht.

Algorithmus 14: getConflictPairs(G)
Bestimmung von Konfliktpaaren

input : BipartiteGraph $G := (T, B; P, R)$
output : **return** Menge der Konfliktpaare *conflictPairs*

```
1  conflictPairs := ∅;
2  andJoins := getANDJoins(G);
3  xorSplits := getXORSplits(G);
4  foreach currentXorSplit ∈ xorSplits do
5  |  foreach currentAndJoin ∈ andJoins do
6  |  |  if |getNodeDisjointPaths (currentXorSplit, currentAndJoin)| > 1
   |  |  then conflictPairs ∪ {(currentXorSplit, currentAndJoin)}
7  |  end
8  end
9  xorJoins := getXORJoins(G);
10 andSplits := getANDSplits(G);
11 foreach currentAndSplit ∈ andSplits do
12 |  foreach currentXorJoin ∈ xorJoins do
13 |  |  if |getNodeDisjointPaths (currentAndSplit, currentXorJoin)| > 1
   |  |  then
   |  |  conflictPairs := conflictPairs ∪ {(currentAndSplit, currentXorJoin)}
14 |  end
15 end
16 return conflictPairs
```

Beispiel: Abbildung 3.13 zeigt das Ergebnis für einen auf Basis von G modifizierten Graphen G'. Das Konfliktpaar $(3, e)$ wurde ermittelt, da die beiden knotendisjunkten Wege $w_1 =< 3, c, 5, e >$ und $w_2 =< 3, d, 7, e >$ zwischen der Entscheidung $xorSplit = 3$ und $andJoin = e$ existieren. Konfliktpaar $(a, 5)$ begründet sich durch die knotendisjunkten Wege $w_3 =< a, 2, b, 5 >$ und $w_4 =< a, 3, c, 5 >$ zwischen der Asynchronisation $andSplit = a$ und dem Kontakt $xorJoin = 5$ jeweils im Graphen G'. Graph G' ist nicht konfliktfrei und somit nicht *well handled*. Die Aktivität e kann nicht ausgeführt werden, falls die Wahl bei Entscheidungsknoten 3 auf die Folgeaktivität c fällt und somit die Transition 7 nicht erreicht wird. Die Transitionen 5 und 7 müssen jedoch für die Ausführbarkeit von e in jedem der möglichen Abläufe erreichbar sein.

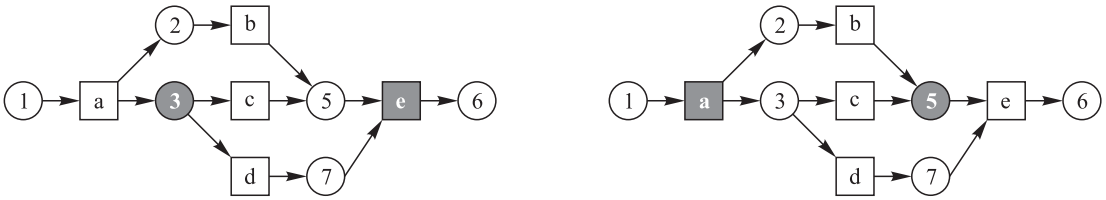


Abbildung 3.13: Konfliktpaare $\{(3, e), (a, 5)\}$ für G'

3.4.5 Bestimmung von Instanzgraphen

Ein Workflowgraph kann ausgehend von einer Starttransition in eine Menge von gültigen Instanzgraphen zerlegt werden, sofern der Graph *well handled* (siehe Kapitel 3.4.4) ist. Die Menge der Instanzgraphen beschreibt alle möglichen Abläufe in einem Workflowgraphen. Dabei entspricht die Anzahl der Instanzgraphen der Anzahl der im Workflowgraphen enthaltenen Entscheidungsknoten (XOR-Splits) plus eins. Enthält ein Workflowgraph keine Entscheidungen, gibt es nur einen möglichen Ablauf, der dem weiteren Verlauf des Workflowgraphen ab der gegebenen Starttransition entspricht. Ein Instanzgraph stellt einen der möglichen Abläufe innerhalb des Graphen ausgehend von einer gegebenen Starttransition dar. Die Bestimmung von Instanzgraphen erfolgt nach Algorithmus 15.

Beschreibung Algorithmus 15: Der Algorithmus bestimmt die Menge aller Instanzgraphen *instGraphs* des Graphen G ausgehend von einem Startknoten *startNode*. Es wird ein neuer Instanzgraph *instGraph* erzeugt und mit dem Ausgangsknoten *startNode* initialisiert. Ab hier werden die nachfolgenden Schritte ausgeführt:

1. Trage alle Instanzgraphen *graphsToRemove* aus der Menge *instGraps* aus, die entfernt werden müssen und alle neu begonnenen Instanzgraphen *pathsToAdd* in die Menge der Instanzgraphen *instGraphs* ein, die im vorangestellten Durchlauf bestimmt wurden.

2. Bestimme den nächsten zu betrachtenden Instanzgraphen *currenInstGraph*. Wurde über alle Elemente der Menge *instGraphs* iteriert, dann fahre fort mit Schritt 7.

3. Bestimme den nächsten Knoten *currentInstGraphNode* im zu betrachtenden Instanzgraphen *currentInstGraph*. Wenn für den Knoten *currentInstGraphNode*

Algorithmus 15: getInstanceGraphs(G, *startNode*)
Bestimmung der Menge von Instanzgraphen

input : BipartiteGraph $G := (T, B; P, R)$, $startNode \in \{T \cup B\}$
output : Menge der Instanzgraphen *instGraphs* ab *startNode*

```
1  instGraph := ∅;
2  new instGraph := (T_S, B_S; P_S, R_S);
3  if startNode ∈ T then T_S := T_S ∪ {startNode} else B_S := B_S ∪ {startNode};
4  instGraphs := instGraphs ∪ {instGraph};
5  graphsToAdd := ∅; graphsToRemove := ∅; changed := false;
6  repeat
7      instGraphs := instGraphs ∪ {graphsToAdd};
8      graphsToAdd := ∅; instGraphs := instGraphs \ {graphsToRemove};
9      graphsToRemove := ∅; changed := false;
10     foreach currentInstGraph(T_I, B_I; P_I, R_I) ∈ instGraphs do
11         foreach currentInstGrapheNode ∈ currentInstGraph do
12             if g_N(currentInstGraphNode) = 0 mit g_N(x) auf currentInstGraph then
13                 successors := g_N(currentInstGraphNode) mit g_N(x) auf G;
14                 if currentInstGraphNode ∈ T then
15                     foreach currentNode ∈ successors do
16                         T_I := T_I ∪ {currentSuccessor};
17                         R_I := R_I := R_I ∪ {(currentInstGraphNode, currentNode)};
18                         changed := true;
19                     end
20                 end
21                 else
22                     if |successors| > 1 then
23                         graphsToRemove ∪ currentInstGraph;
24                         foreach currentSuccessor ∈ successors do
25                             new copyGraph := (T_C, B_C; P_C, R_C);
26                             copyGraph := (T_C ∪ T_I, B_C ∪ B_I; P_C ∪ P_I, R_C ∪ R_I);
27                             B_C := B_C ∪ {currentSuccessor};
28                             P_C := P_C ∪ {(currentInstGraphNode, currentSuccessor)};
29                             grapphsToAdd := grapphsToAdd ∪ {copyGraph};
30                             changed := true;
31                         end
32                     end
33                     else
34                         foreach currentSuccessor ∈ successors do
35                             B_C := B_I ∪ currentSuccessor;
36                             P_C := P_C ∪ {(currentInstGraphNode, currentSuccessor)};
37                             changed := true;
38                         end
39                     end
40                 end
41             end
42         end
43     end
44 until changed;
45 return instGraphs
```

noch keine Nachfolger im momentan betrachteten Instanzgraphen *currentInstGraph* existieren, fahre fort mit Schritt 5, sonst weiter mit Schritt 3.

4. Ermittle die Nachfolger *successors* von *currentInstGraphNode* im Ausgangsgraphen G. Prüfe für jeden Nachfolger *currentNode* der Menge *successors*, ob er Element der Menge der Aktivitäten T des Ausgangsgraphen ist. Wenn nicht, weiter mit Schritt 5. Ist $currentNode \in T$ kann es sich nicht um einen Entscheidungsknoten handeln. Es entsteht somit kein neuer Instanzgraph. Alle Nachfolger von *currentNode* werden in die Menge T_I und alle Kanten (*currentInstGraphNode*, *currentNode*) in die Menge R_I des momentan betrachteten Instanzgraphen *currentInstGraph* eingetragen. Der aktuelle Durchlauf wird mit $changed := true$ markiert. Weiter mit Schritt 3.

5. Knoten *currentInstNode* ist Element der Menge der Transitionen B. Es wird geprüft, ob der Knoten *currentInstNode* ein Entscheidungsknoten ist. Dies ist der Fall, wenn er mehr als einen Nachfolger hat. Hat er mehr als einen Nachfolger (sonst weiter mit Schritt 6), erhöht sich die Anzahl der Instanzgraphen um die Anzahl der Nachfolger minus eins. *currentInstGraph* wird als zu entfernender Graph der Menge *graphsToRemove* zugeordnet. Anschließend wird für jeden Nachfolger *currentSuccessor* ein neuer Graph $copyGraph(T_C, B_C; P_C, R_C)$ erzeugt und mit den Mengen aller Knoten und Kanten von *currentInstGraph* initialisiert. Zusätzlich wird der Knoten *currentSuccessor* der Menge B_C und die Kante (*currentInstGraphNode*, *currentSuccessor*) der Menge P_C zugeordnet. Es folgt das Eintragen von *copyGraph* in die Menge *graphsToAdd*. Der aktuelle Durchlauf wird mit $changed = true$ markiert. Weiter mit Schritt 3.

6. Es gibt demnach nur einen oder gar keinen Nachfolger *currentSuccessor*. Gibt es einen, so wird dieser der Menge B_I und die Kante (*currentInstGraphNode*, *currentSuccessor*) der Menge P_C zugeordnet und der aktuelle Durchlauf wird mit $changed = true$ markiert. Weiter mit mit Schritt 3.

7. Es wurde über alle zu betrachtenden Instanzgraphen der Menge *instGraphs* iteriert. Wurde im aktuellen Durchlauf eine Veränderung in der Menge der Instanzgraphen *instGraphs* oder in den darin enthaltenen Instanzgraphen vorgenommen, so wurde *changed* mit dem Wert *true* belegt. Wenn ja, beginnt ein neuer Durchlauf - weiter mit Schritt 1. Sonst weiter mit Schritt 8.

8. Alle Instanzgraphen des Graphen G ausgehend vom Knoten *startNode* wurden bestimmt. *instGraphs* ist die Ergebnismenge und wird zurückgeliefert.

Beispiel: Abbildung 3.14 zeigt die Menge der durch Algorithmus 15 bestimmten Instanzgraphen ausgehend von Knoten 1.

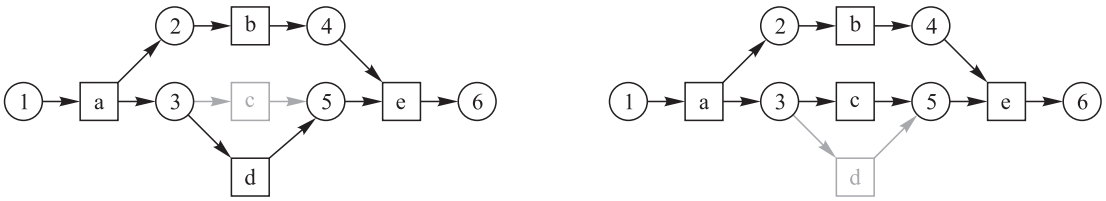


Abbildung 3.14: Menge der Instanzgraphen für G ausgehend von Knoten $startNode = 1$

4 Generierung von Ablaufalternativen

Dieses Kapitel stellt das modellorientierte Konzept für das automatisierte Generieren von Bauabläufen auf Basis eines BIM vor. Der Modellentwurf erfolgt im Hinblick auf die Umsetzung des Konzeptes in Kapitel 5 objektorientiert. Die formale Beschreibung des Modells erfolgt auf Basis der Mengen- und Relationenalgebra und stützt sich dabei auf die in Kapitel 3 vorgestellten Methoden und Algorithmen.

4.1 Systementwurf

Auf Basis der in den Kapiteln 2.5.1 und 2.5.2 erläuterten Abhängigkeiten sowie dem Automatisierungsansatz nach Kapitel 2.5.3 wird ein Modell entwickelt, das alle für diesen Ansatz notwendigen Informationen beinhaltet. Ziel ist die formale Abbildung der subjektiven Entscheidungsprozesse der Bauablaufplanung. Das Modell soll es erlauben, die Teilabläufe in eine logische Reihenfolge zu bringen und den subjektiven Entscheidungsweg, der zur Bildung eines Bauablaufs geführt hat, in ein Softwaresystem zu übertragen. Der Vorteil besteht darin, dass dieser Entscheidungsweg formal abgebildet wird und dadurch dessen Wiederverwendbarkeit ermöglicht wird. Wenn beispielsweise ein Vorgang für die Erstellung einer Wand modelliert wurde, dann ist er auch zukünftig durch dessen Abbildung im Modell vorhanden.
Abbildung 4.1 zeigt die Prinzipskizze des angestrebten Systementwurfs[1] ausgehend von einem der Planung entstammenden Bauwerksinformationsmodell.
Die innerhalb des Systems auszuführenden Arbeitsschritte ergeben sich dabei wie folgt:

1. **Strukturierungsphase:** Es erfolgt die Definition einer dem Bauwerksinformationsmodell übergeordneten Struktur in Form von Struktureinheiten. Die Definition von Struktureinheiten basiert dabei auf der Struktur des Bauwerks, vorgegebenen Bauabschnitten als auch Gewerken und entspricht in diesem Sinne einer Grobterminplanung. Eine Struktureinheit beschreibt alle darin enthaltenen Bauelemente, die in einem zugehörigen Teilablauf herzustellen sind. Die Reihenfolge der Erstellung von Struktureinheiten und somit deren Teilabläufe ist dabei durch den Anwender vorzugeben. Diese Vorgabe ist notwendig, da nicht automatisiert auf die richtige Reihenfolge dieser Teilabläufe geschlossen werden kann. Beispielsweise ist die Vorgehensweise, ein Bauwerk beginnend mit dem Fundament bis zum Dachgeschoss herzustellen, zwar in den meisten Fällen die maßgebende, jedoch

[1] vgl. [Claus u. Schwill 2003] Als System wird in der Informationstechnologie die Zusammenfassung mehrerer Komponenten zu einer Einheit für ein wohldefiniertes Bündel von Aufgaben verstanden - hier die Gesamtlösung für die automatisierte Generierung von Bauablaufplänen.

ist sie nicht zwingend richtig. Als Beispiel kann der Umbau eines Bauwerks infolge einer Umnutzung herangezogen werden, bei dem aufgrund nutzungsbedingter Vorgaben die Umbauten in der Reihenfolge zwanzigstes, fünfzehntes, siebzehntes Obergeschoss ausgeführt werden müssen.

Es werden alle für den jeweils zu detaillierenden Teilablauf relevanten Bauteile aufgrund der Zugehörigkeit zu ihren Struktureinheiten aus dem BIM extrahiert und in Form definierter Bauteilobjekte in das System überführt.

2. **Generierungsphase:** Es folgt die automatisierte Zuordnung von Vorgängen für jedes der in Schritt 1 extrahierten Bauteilobjekte einer jeden Struktureinheit. Dafür werden Vorgänge innerhalb der Datenbasis gesucht, die als Ergebnis ein passendes Bauteilobjekt liefern, und somit automatisch bestimmt. Können keine passenden Vorgänge gefunden werden, muss zumindest ein passender Vorgang vom Anwender definiert und in die Datenbasis eingefügt werden. Wenn für jedes zu betrachtende Bauteilobjekt ein Vorgang bestimmt wurde, wird auf Basis der gefundenen Vorgänge ein möglicher Teilablauf generiert. Dies erfolgt durch die Verknüpfung der Resultate von Vorgängen mit identischen Vorbedingungen nach These 5.

3. **Kopplungsphase:** Nach der Bestimmung zumindest eines gültigen Teilablaufs für jede Struktureinheit erfolgt deren Kopplung zu einem Gesamtablauf auf Basis der übergeordneten Grobterminplanung aus Punkt 1.

Gleiche Ausgangsinformationen führen dabei im Idealfall zu gleichen Ergebnissen auf Basis des in der Datenbasis vorgehaltenen Wissens. Der Automatisierungsansatz führt dadurch zu reproduzierbaren Ergebnissen. Durch die formale Abbildung erfolgt eine eingeschränkte Entkopplung von subjektiven Entscheidungen des Planers. Einschränkend muss festgestellt werden, dass für den Bauprozess viele Besonderheiten aufgrund des Unikatcharakters von Bauwerken und aufgrund von äußeren Bedingungen existieren. Auch ist die Kombination von Teilabläufen nicht völlig frei, da zwischen ihnen ebenfalls Abhängigkeiten bestehen, die sich abhängig vom Bauwerk unterscheiden und vom Planer berücksichtigt werden müssen. Aus diesem Grund besteht die Notwendigkeit des manuellen Eingriffs. An dieser Stelle kommt die subjektive Entscheidung des Planers wieder in das System.
Der vormals manuelle Prozess der Bauablaufplanung wird auf Basis eines neu entwickelten Modells und darauf operierender Methoden in einen teilautomatisierten Prozess überführt. Ziel der Arbeit ist demnach die Entwicklung eines den Planer unterstützenden Werkzeugs für die Bauablaufplanung.

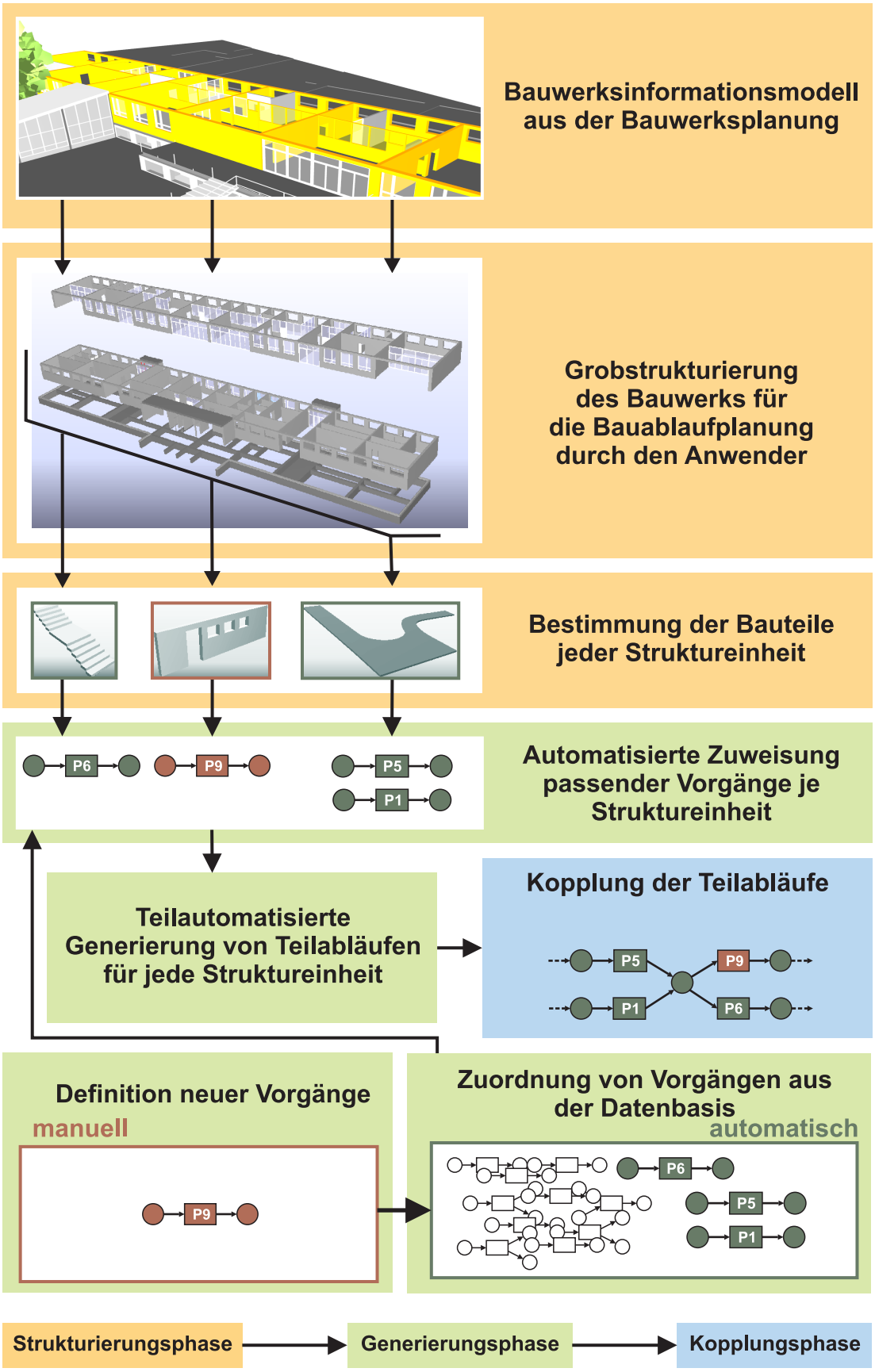


Abbildung 4.1: Prinzipskizze des Systementwurfs

4.2 Modellbildung

Unter Modellbildung wird die hinreichend genaue Abbildung eines Systems für ein konkretes Problem verstanden. Das System innerhalb dieser Arbeit sind Methoden für das teilautomatisierte Generieren von Bauabläufen, die zur Herstellung von Bauwerksabschnitten beziehungsweise kompletten Bauwerken führen.
Innerhalb des vorliegenden Kapitels erfolgt die Entwicklung eines Modells für den gewählten Lösungsansatz nach Kapitel 2.5 und des Systementwurfs gemäß dem vorangestellten Kapitel 4.1.

4.2.1 Digitales Bauwerksinformationsmodell

Für die Abbildung des BIM erfolgt keine eigens definierte, formale Abbildung. Prinzipiell ist der angestrebte Systementwurf für jede Form eines digitalen BIM anwendbar. Es gelten jedoch nachfolgende Mindestanforderungen:

Anforderungen

Die im BIM enthaltenen Informationen müssen in den Modellentwurf, der in den nachfolgenden Kapiteln entworfen wird, überführbar sein. Um dies zu gewährleisten, müssen folgende Kriterien zwingend erfüllt sein:

1. **Alle Objekte eines BIM und somit auch die Beschreibung von Bauteilen unterliegen einer Taxonomie.** Das heißt, alle Elemente des BIM sind jeweils einer Klasse von Objekten zugeordnet oder es ist möglich, sie einer Klasse eindeutig zuzuordnen. Beispielsweise gehören Wände, Türen oder Fundamente jeweils einer Klasse von Objekten an oder können beispielsweise auf Basis einer textuellen oder numerischen Bezeichnung zugeordnet werden. Diese Voraussetzung ergibt sich aus dem notwendigen Vergleich von Bauelementen für die Bestimmung der Ähnlichkeit, da nur der Vergleich identischer Klassen an dieser Stelle zu sinnvollen Ergebnissen führt.

2. **Alle Bauteile des BIM müssen eindeutig und persistent identifizierbar sein.** Dies ist sowohl für die Eindeutigkeit einer Zuordnung von Vorgängen zu Bauteilen des BIM als auch für die Zuordnung von Bauteilen zu Struktureinheiten des Gebäudemodells zwingend erforderlich.

3. **Der Bauteilbeschreibung des BIM ist eine Materialbeschreibung zugeordnet.** Das wichtigste Kriterium für die Auswahl der Art und Weise der Herstellung eines Bauteils ist dessen Material. Für die Bestimmung passender Vorgänge ist diese Information somit von entscheidender Bedeutung.

4. **Für alle Bauteile des BIM muss eine geometrische Beschreibung vorliegen.** Auch diese Information spielt für die Bestimmung von Vorgängen eine bedeutende Rolle, da aufgrund der Größe eines Bauteils gegebenenfalls auf technologische Notwendigkeiten geschlossen werden kann und diese somit Einfluss auf die Auswahl passender Vorgänge hat. Ebenso ist diese Information für eine

automatisierte Zuordnung der Voraussetzungen eines Vorgangs von entscheidender Bedeutung (siehe Abschnitt „Schritt 2: Zuordnung von Voraussetzungen zu Vorgängen“ auf Seite 84). Weiterhin ist die Visualisierung des Bauwerks sowohl für die Handhabbarkeit des Systems innerhalb der Strukturierungs- und Generierungsphase als auch für die Kontrolle des generierten Bauablaufs durch den Anwender erforderlich.

4.2.2 Abbildung von Bauteilen

Bauteile werden innerhalb dieser Arbeit anhand eines logischen Datensatzes beschrieben. Die Art der Abbildung folgt dem Ansatz der Objektorientierung. Ein Bauteil wird demnach als Objekt betrachtet und mittels seiner Eigenschaften, bezogen auf das Problem der teilautomatisierten Generierung von Bauablaufplänen, hinreichend genau beschrieben. Der notwendige Detailgrad und die Art der Abbildung wird aus den Anforderungen des Systementwurfs nach Abbildung 4.1 abgeleitet.

Anforderungen

Ziel der Abbildung von Bauteilen ist die Realisierung einer formalen Zuordnung von Vorgängen, die zur Erstellung des jeweiligen Bauteils führen oder als Voraussetzung des Vorgangs anzusehen sind. Zu diesem Zweck werden Vorgänge mit ihren Voraussetzungen und deren Resultat definiert und innerhalb einer Datenbasis gespeichert. Voraussetzungen als auch das Resultat eines Vorgangs werden jeweils als ein bauteilbeschreibendes Objekt abgebildet (vgl. Kapitel 4.2.4 und 4.2.5). Die Zuordnung eines Vorgangs aus der Domäne der Datenbasis zu einem Bauteilobjekt eines BIM erfolgt, wenn das Bauteilobjekt des BIM und das Ergebnis des betrachteten Vorgangs einander ähnlich sind. Abbildung 4.2 stellt den Vorgang der Zuordnung von Bauteilobjekten als Ergebnis von Vorgängen als Prinzipskizze dar.[2]
Im Wesentlichen erfolgt die Zuordnung von Vorgängen zu Bauteilobjekten eines BIM, wenn das Ergebnis eines Vorgangs in der Datenbasis dem Bauteilobjekt des BIM in hohem Maße ähnlich ist. Grundlage hierfür ist die Implikation, dass ein Vorgang, der ein Bauteil erzeugt, auf die Erzeugung eines ähnlichen Bauteils übertragen werden kann.
Im Grundsatz basiert die Bestimmung der Ähnlichkeit zweier Objekte auf dem Vergleich ihrer Eigenschaften. Je mehr Eigenschaften übereinstimmen, umso ähnlicher sind sie sich. Je detaillierter ein Bauteil demnach mittels seiner Eigenschaften beschrieben wird, umso höher ist die Menge vergleichbarer Kriterien und somit die erreichbare Qualität des Vergleichs. Gleichzeitig steht die Anzahl der beschreibenden Eigenschaften im umgekehrten Verhältnis zum notwendigen Definitionsaufwand. An dieser Stelle muss demnach entweder der Aufwand für die Definition der Eigenschaften reduziert oder aber

[2] Die Identifikation von Bauteilobjekten, die Voraussetzungen eines Vorgangs darstellen, erfolgt in der ersten Stufe ähnlich der Methode zur Bestimmung von Ergebnissen eines Vorgangs. Auf die Art der Abbildung von Bauteilobjekten hat dies keinen Einfluss. Aus diesem Grund wird an dieser Stelle auf eine Erklärung verzichtet. Beide Methoden werden jedoch in Kapitel 4.3.2 ausführlich erläutert.

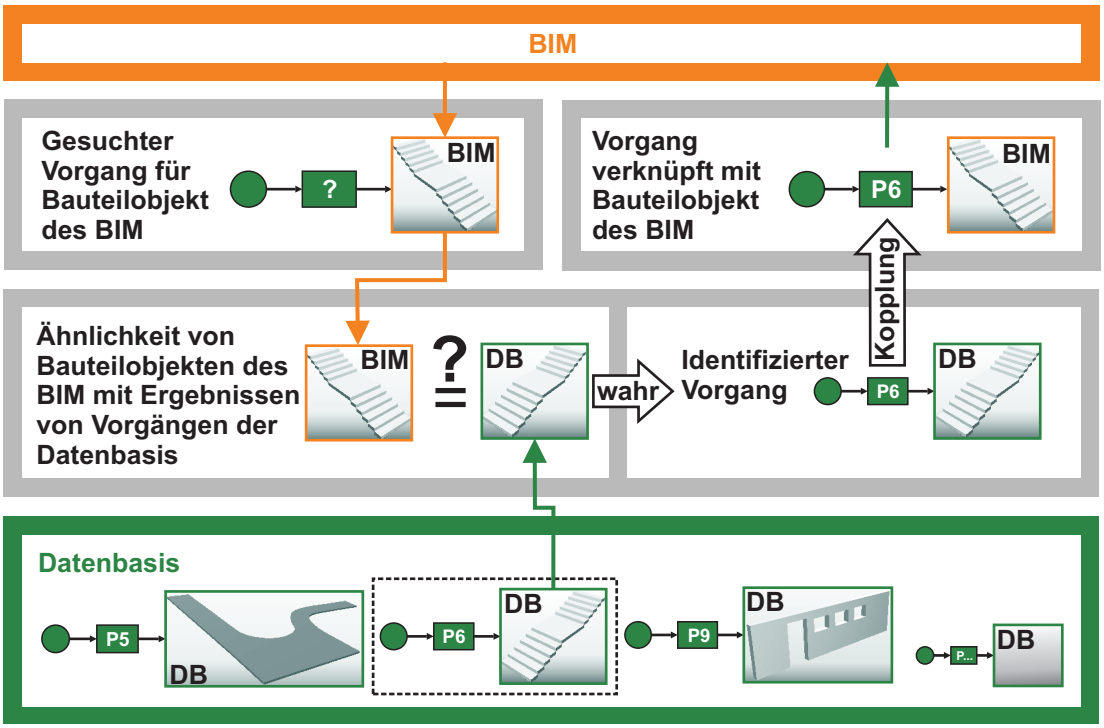


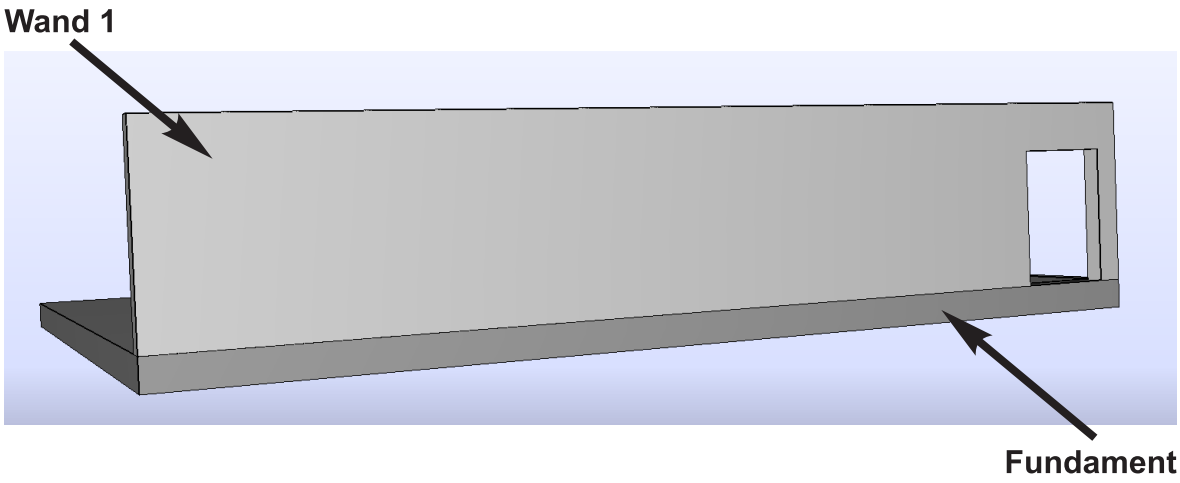
Abbildung 4.2: Prinzip des Vergleichs und der Zuordnung von Bauteilobjekten als Ergebnis von Vorgängen

eine Einschränkung hinsichtlich der Menge abzubildender Eigenschaften vorgenommen werden. Eine weitere Möglichkeit ist eine generische Art der Abbildung von Attributen und deren zugehöriger Werte. Dabei werden die Attribute eines Objekts nicht innerhalb deren Klassenbeschreibung, sondern zur Laufzeit[3] der Applikation definiert. Objekte können dadurch in nahezu beliebiger Ausprägung und Detailtreue beschrieben werden. Für die Abbildung der Bauteilinformationen wurde ein Ansatz gewählt, der dem generischen Abbildungsansatz folgt. Er erlaubt das Beschreiben und Vergleichen von Objekten auf verschiedenen Detailstufen und bietet somit die größtmögliche Flexibilität des angestrebten Modells. Die Bestimmung der Ähnlichkeit wird in Kapitel 4.2.3 näher erläutert. Für die Abbildung der Bauteile bedeutet dies, dass alle Attribute hierarchisch in Form einer Baumstruktur abgebildet werden. Abbildung 4.3 zeigt als Beispiel die mögliche Modellierung der Datenstrukturen für eine Bodenplatte und eine Wand.

Formale Abbildung

Formal wird die Menge aller Bauelemente B als Menge von Objekten abgebildet. Jedes Bauelement $b \in B$ besteht aus einer endlichen Menge von Attributen a. Attribute spiegeln die Eigenschaften eines Objekts der Realität wider. Sie drücken den Status eines Bauteils zum Zeitpunkt der Betrachtung aus.

[3]Laufzeit - bezeichnet in diesem Zusammenhang die Phase während der Ausführung eines Programms oder Algorithmus



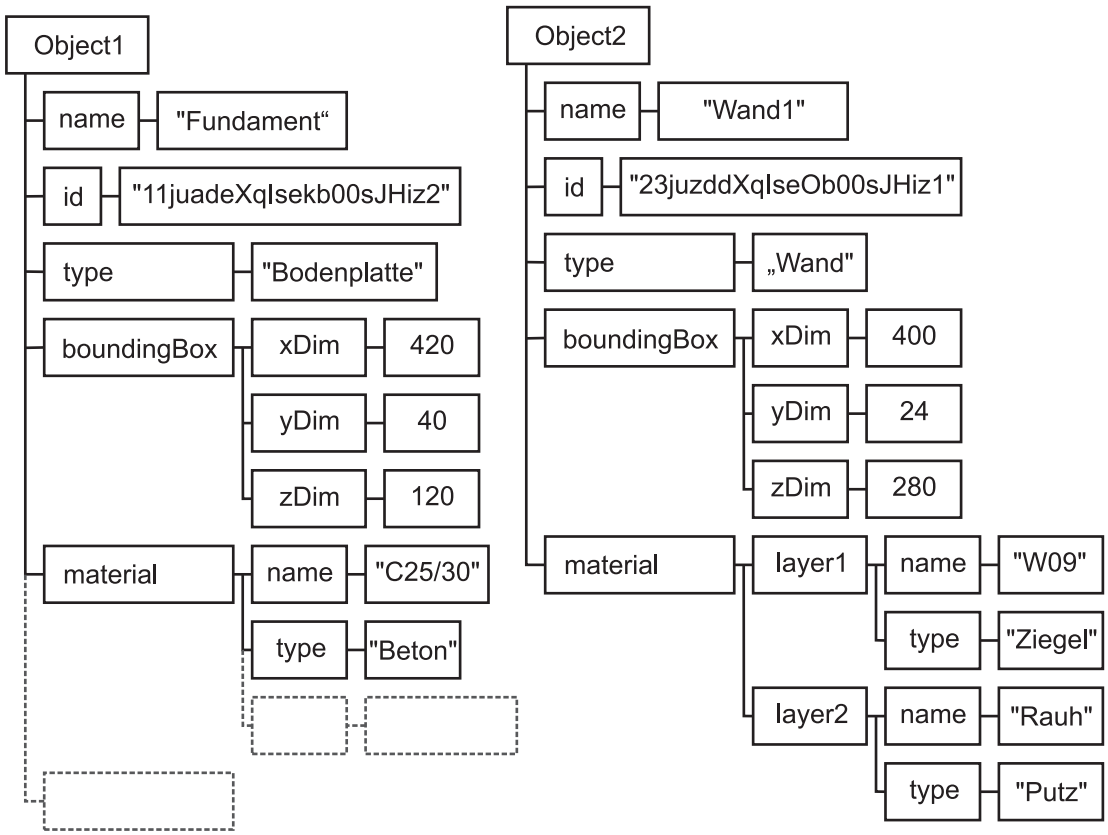


Abbildung 4.3: Mögliche generische Abbildung der Eigenschaften eines Bauteils am Beispiel eines Fundaments und einer Wand

$$B := \{b_1, \ldots, b_i \mid b_i \text{ ist ein Bauelement}\} \text{ mit } i \in \mathbb{N}$$
$$b_i := \{a_1 \ldots a_j \mid a_j \text{ ist ein Attribut }\} \text{ mit } j \in \mathbb{N} \tag{4.1}$$

Teilmengen Innerhalb der Domäne existieren Teilmengen von Bauteilen. Es handelt sich um die Menge der Bauteile des betrachteten BIM B_{BIM}, die Menge der Bauteile der Datenbasis B_{DB} sowie die Menge von Bauteilen der jeweiligen Struktureinheit des BIM B_{su} (siehe Kapitel 4.2.6).

$$B_{BIM} := \{b_1, \ldots, b_i \mid b_i \text{ ist Bauelement des BIM}\} \text{ mit } i \in \mathbb{N}$$
$$B_{su} := \{b_1, \ldots, b_k \mid b_k \text{ ist Bauelement der Struktureinheit } su\} \text{ mit } j, k \in \mathbb{N}$$
$$B_{DB} := \{b_1, \ldots, b_n \mid b_n \text{ ist Bauelement der Datenbasis}\} \text{ mit } n \in \mathbb{N}$$

$$\text{mit } B := B_{BIM} \cup B_{DB}$$
$$\text{und } B_{su} \subset B_{BIM} \tag{4.2}$$

Attribut Ein Attribut ist ein geordnetes Tripel[4] bestehend aus dem Attributtyp a_t, dem Namen a_n und dem Wert a_v des Attributs. Der Name a_n eines Attributs muss innerhalb des Objekts eindeutig identifizierbar sein. Er darf demnach nicht mehrfach vergeben werden. Das Zeichen "." kennzeichnet innerhalb des Namens a_n ausschließlich das Trennzeichen der Baumebene (vgl. Beispiel auf Seite 66).

$$\begin{aligned} A := (a_t, a_n, a_v) \;& \text{mit } a_t \in AT \text{ - Typ des Attributs nach Formel 4.4} \\ & \text{mit } a_n := (c_1, \ldots, c_i) \mid c_i \in \mathbb{T} \text{ - ein eindeutiger Attributname im Objekt} \\ & \quad \text{es gilt} \bigwedge_{a_j \in b_i, a_k \in b_i} a_{j_n} \neq a_{k_n} \quad \text{für } j \neq k \\ & \quad \text{und} \bigwedge_{c_i \in a_n} c_i = \text{"."} \mid c_i \text{ ist Trennzeichen der Baumebene} \\ & \text{und } a_v \in a_t \text{ - der Wert des Attributs (engl. value)} \end{aligned} \tag{4.3}$$

Wertmenge Der Wert eines Attributs a_v wird durch den Attributtyp $a_t \in AT$ in seinem für ihn gültigen Wertebereich eingeschränkt. Formel 4.4 kennzeichnet die verwendeten Wertebereiche. Sie umfassen die Menge der ganzen Zahlen $\mathbb{Z}$, der rationalen Zahlen $\mathbb{Q}$, Wahrheitswerte $\mathbb{W}$, alphanumerische Werte $\mathbb{T}$ und den Wertebereich *ID* für die Darstellung eines eindeutigen Identifikators in Anlehnung an den GUID der IFC[5].

[4] auch 3-Tupel - geordnete Aufzählung von drei Objekten, siehe [Scheid 2000]

[5] Der GUID (Globaly Unique Identifier) der IFC besteht aus 128 Bit und wird durch 22 gültige Zeichen der Menge *ID* aus Formel 4.4 ausgedrückt. Siehe: http://www.iai-tech.org/ifc/IFC2x3/TC1/html/ifcutilityresource/lexical/ifcgloballyuniqueid.htm Abruf: 15.11.2010

$$AT := \{\mathbb{Z}, \mathbb{Q}, \mathbb{W}, \mathbb{T}, ID\}$$

mit $\mathbb{Z}$ ist die Menge der ganzen Zahlen $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
mit $\mathbb{Q}$ ist die Menge der rationalen Zahlen $\mathbb{Q} = \{p/q \mid p, q \in \mathbb{Z} \wedge q \neq 0\}$
mit $\mathbb{W}$ ist ein Wahrheitswert $W = \{true, false\}$
mit $\mathbb{T}$ ist Text
und $ID := (c_1, \ldots, c_i, \ldots, c_{22}) \mid c_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F,$
$G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d,$
$e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, \_, \$\}$ (4.4)

Identifikator Objekte müssen eindeutig und persistent identifizierbar sein. Sie verfügen aus diesem Grund über einen eindeutigen Identifikator. Der Identifikator wird als Attribut des Objekts mit dem Namen $a_n =$"id" und dem Attributtyp $a_t = ID$ abgebildet. Der Wert a_v des Attributs mit dem Namen $a_n =$"id"wird durch w_{id} bezeichnet und bezeichnet den vorausgesetzten Identifikator eines Bauteils des BIM (vgl. [Tulke 2010] oder [Hanff 2003]).

$$\bigwedge_{B} \bigvee_{a_i \in B} a_{n_i} = \text{"}id\text{"} \wedge a_{v_i} = w_{id}$$
$$\bigwedge_{B_i, B_j} w_{id_i} \neq w_{id_j} \quad \text{für } i \neq j \tag{4.5}$$

Typ Bauteile müssen für einen effizienten Vergleich einer Klasse von Objekten angehören (vgl. Abschnitt „Anforderungen“ auf Seite 61). Der Bauteiltyp wird als Attribut des Objekts mit dem Namen $a_n =$"$type$" und dem Attributtyp $a_t = \mathbb{T}$ abgebildet.

$$\bigwedge_{B} \bigvee_{a_i \in B} a_{n_i} = \text{"}type\text{"} \wedge a_t = \mathbb{T} \tag{4.6}$$

Material Für die Bestimmung von passenden Vorgängen ist die Angabe eines Materials zwingend notwendig (vgl. Abschnitt „Anforderungen“ auf Seite 61). Das Material eines Bauteils kann in verschiedenen Ausprägungen vorliegen. Für den angestrebten Lösungsansatz werden sowohl Bauteile mit einschichtigem als auch mehrschichtigem Materialaufbau unterstützt. Vorausgesetzt wird jeweils der Typ und Name eines Materials. Die Attributnamen a_n setzen sich dementsprechend zusammen und werden mit $material.layer*.type$ und $material.layer*.name$ bezeichnet. Das Zeichen $*$ steht dabei stellvertretend für eine Zahl $n \in \mathbb{Z}$, welche die Nummer der Materialschicht repräsentiert. Dabei wird die Angabe mindestens eines Materials vorausgesetzt, welches den Attributen $material.layer1.type$ und $material.layer1.name$ entspricht.

$$\bigwedge_{B} \bigvee_{a_i \in B} a_{n_i} = \text{"}material.layer1.type\text{"} \wedge a_t = \mathbb{T}$$
$$\bigwedge_{B} \bigvee_{a_i \in B} a_{n_i} = \text{"}material.layer1.name\text{"} \wedge a_t = \mathbb{T} \tag{4.7}$$

Bounding-Box Für die Bestimmung passender Vorgänge und für die Identifikation von Voraussetzungen eines Vorgangs ist die Angabe der Geometrie eines Bauteils zwingend notwendig (vgl. Abschnitt „Anforderungen“ auf Seite 61). Als Mindestanforderung gilt die Angabe der Bounding-Box[6]. Die Bounding-Box eines Bauteils wird als Gruppe von Attributen des Objekts mit dem Namen $a_n =$"$boundingBox.xDim$", $a_n =$"$boundingBox.yDim$" und $a_n =$"$boundingBox.zDim$" mit dem Attributtyp $a_t = \mathbb{Q}$ abgebildet. Die Angaben ihrer Ausdehnung sind bezogen auf den Ursprung eines lokalen Koordinatensystems. Die numerischen Werte entsprechen der Angabe in Meter.

$$\bigwedge_{B} \bigvee_{a_i \in B} a_{n_i} = \text{"}boundingBox.xDim\text{"} \wedge a_t = \mathbb{Q}$$
$$\bigwedge_{B} \bigvee_{a_i \in B} a_{n_i} = \text{"}boundingBox.yDim\text{"} \wedge a_t = \mathbb{Q}$$
$$\bigwedge_{B} \bigvee_{a_i \in B} a_{n_i} = \text{"}boundingBox.zDim\text{"} \wedge a_t = \mathbb{Q} \tag{4.8}$$

Beispiel: Das nachfolgende Beispiel zeigt die formale Abbildung der Objekte "Wand1“ und "Fundament"nach Abbildung 4.3. Die hierarchische Ordnung fließt in Form der Namensgebung der Attribute ein, indem die Ebenen der Baumstruktur entsprechend der Definition durch einen Punkt getrennt werden.

$b_1 := \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9\}$ $\quad a_0 := (\mathbb{T}, \text{"}name\text{"}, \text{"}Wand1\text{"})$

$a_1 := (ID, \text{"}id\text{"}, \text{"}23juzddXqIseOb00sJHiz1\text{"})$ $\quad a_2 := (\mathbb{T}, \text{"}type\text{"}, \text{"}Wand\text{"})$
$a_3 := (\mathbb{Q}, \text{"}boundingBox.xDim\text{"}, 4.0)$ $\quad a_4 := (\mathbb{Q}, \text{"}boundingBox.yDim\text{"}, 0.24)$
$a_5 := (\mathbb{Q}, \text{"}boundingBox.zDim\text{"}, 2.8)$ $\quad a_6 := (\mathbb{T}, \text{"}material.layer1.name\text{"}, \text{"}W09\text{"})$
$a_7 := (\mathbb{T}, \text{"}material.layer1.type\text{"}, \text{"}Ziegel\text{"})$ $\quad a_8 := (\mathbb{T}, \text{"}material.layer2.name\text{"}, \text{"}Putz\text{"})$
$a_9 := (\mathbb{T}, \text{"}material.layer1.type\text{"}, \text{"}Rauh\text{"})$

$b_2 := \{a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}, a_{17}\}$ $\quad a_{10} := (\mathbb{T}, \text{"}name\text{"}, \text{"}Fundament\text{"})$

$a_{11} := (ID, \text{"}id\text{"}, \text{"}11juadeXqIsekb00sJHiz2\text{"})$ $\quad a_{12} := (\mathbb{T}, \text{"}type\text{"}, \text{"}Bodenplatte\text{"})$

[6] Beschreibt einen Quader mit minimalem Volumen, der das zugehörige Objekt vollständig umschließt. Eine Bounding-Box wird in der Regel achsenparallel angegeben.

$a_{13} := (\mathbb{Q}, "boundingBox.xDim", 4.2)$ $a_{14} := (\mathbb{Q}, "boundingBox.yDim", 0.4)$
$a_{15} := (\mathbb{Q}, "boundingBox.zDim", 1.2)$ $a_{16} := (\mathbb{T}, "material.name", "C25/30")$
$a_{17} := (\mathbb{T}, "material.type", "Beton")$

4.2.3 Ähnlichkeit von Bauteilen

Um die Aussage der Ähnlichkeit zweier Objekte formal treffen zu können, ist die Definition eines Ähnlichkeitsmaßes erforderlich. Das Ähnlichkeitsmaß stellt die Ähnlichkeit miteinander verglichener Bauteilobjekte als normalisierte Zahl im Intervall $[0, 1]$ dar. Die Zahl 1 steht dabei für die größtmögliche Ähnlichkeit und die Zahl 0 für die größtmögliche Verschiedenheit von Objekten.
Das Konzept für die Bestimmung der Ähnlichkeit von Bauteilen, auf Basis deren generischer Beschreibung nach Kapitel 4.2.2, wurde innerhalb eines assoziierten Forschungsprojekts entwickelt (vgl. [Tauscher u. a. 2007; Mikulakova u. a. 2008; Mikulakova 2010; Mikulakova u. a. 2010]).
Die Bestimmung des Ähnlichkeitsmaßes erfolgt dabei in drei Stufen. Es wird ein jeweils normiertes *strukturelles* und ein *inhaltliches Ähnlichkeitsmaß* bestimmt, die in Kombination das *Gesamtähnlichkeitsmaß* ergeben.

Strukturelle Ähnlichkeit Die Bestimmung der strukturellen Ähnlichkeit bezieht sich auf den Vergleich des Vorhandenseins von Attributen eines Bauteilobjekts. Je mehr Attribute beider Bauteilobjekte übereinstimmen, umso höher ist die strukturelle Ähnlichkeit beider Bauteilobjekte. Die Berechnung der strukturellen Ähnlichkeit erfolgt nach dem *Feature Contrast Model* (FCM) nach [Tversky 1977] und wurde wie folgt erweitert:

$$sim_{FCM}(b_i, b_j) := \frac{\alpha \cdot \left|A_{N_i} \cap A_{N_j}\right| - \beta \cdot \left|A_{N_i} \setminus A_{N_j}\right| - \gamma \cdot \left|A_{N_j} \setminus A_{N_i}\right|}{\left|A_{N_i} \cup A_{N_j}\right|}$$

mit α, β, γ als Wichtungen der Terme, es gilt: $\alpha + \beta + \gamma = 1$
mit $b_i \in B$ und $b_j \in B$
mit A_{N_i} als der Menge der Attributnamen a_n von B_i
und A_{N_j} als der Menge der Attributnamen a_n von B_j (4.9)

Dabei drückt $\left|A_{N_i} \cap A_{N_j}\right|$ die Anzahl der in den Bauteilobjekten b_1 und b_2 übereinstimmenden Attribute aus. Attribute stimmen unbeachtet ihrer Werte überein, wenn deren Attributname a_{n_i} und a_{n_j} identisch ist. Durch $\left|A_{N_i} \setminus A_{N_j}\right|$ wird die Anzahl der Attribute, die nicht in b_j enthalten sind und $\left|A_{N_j} \setminus A_{N_i}\right|$ die Anzahl der Attribute, die nicht in b_i enthalten sind, ausgedrückt. $\left|A_{N_i} \cup A_{N_j}\right|$ kennzeichnet die Gesamtmenge der in den zu vergleichenden Bauteilobjekten enthaltenen Attribute. Die Wichtungen α, β und γ drücken die Wichtigkeit des entsprechenden Koeffizienten aus. Als zweckmäßige Werte haben sich innerhalb des assoziierten Forschungsprojekts infolge empirischer Versuche die Werte $\alpha = 0,7$, $\beta = 0,2$ und $\gamma = 0,1$ als zweckmäßig herausgestellt (vgl. [Mikulakova 2010]).

Durch Anwendung von Formel 4.10 wird das Ergebnis aus Formel 4.9 normalisiert.

$$sim_{struct}(b_i, b_j) := \frac{sim_{FCM}(b_i, b_j) + max(\alpha, \beta, \gamma)}{1 + max(\alpha, \beta, \gamma)}$$

sim_{struct} Strukturelles Ähnlichkeitsmaß (engl. Structural Similarity) (4.10)

Es ergibt sich der Wert sim_{struct}, welcher die strukturelle Ähnlichkeit zweier Bauteilobjekte bezogen auf deren übereinstimmende Attribute repräsentiert. Formel 4.10 ist gültig, wenn alle Terme des Zählers größer 0 sind. Ist dies nicht der Fall, gelten für den Wert α folgende Ersetzungen:

$$\begin{aligned}
&\left(\left|A_{N_i} \setminus A_{N_j}\right| = 0\right) \Rightarrow \alpha := \alpha + \beta \\
&\left(\left|A_{N_j} \setminus A_{N_i}\right| = 0\right) \Rightarrow \alpha := \alpha + \gamma \\
&\left(\left|A_{N_i} \setminus A_{N_j}\right| = 0\right) \wedge \left(\left|A_{N_j} \setminus A_{N_i}\right| = 0\right) \Rightarrow \alpha := \alpha + \beta + \gamma
\end{aligned} \tag{4.11}$$

Falls die beiden Bauteilobjekte b_i und b_j keine übereinstimmenden Attribute besitzen, so ergibt sich für das strukturelle Ähnlichkeitsmaß sim_{struct} der Wert 0.

$$\left(\left|A_{N_i} \cap A_{N_j}\right| = 0\right) \Rightarrow sim_{struct}(b_i, b_j) := 0 \tag{4.12}$$

Inhaltliche Ähnlichkeit Die inhaltliche Ähnlichkeit bezieht sich auf den Vergleich der Werte übereinstimmender Attribute zweier Bauteilobjekte. Die Ermittlung der inhaltlichen Ähnlichkeit erfolgt auf Basis von [Richter 2008] für die Anwendung auf beliebige Werte[7] der verallgemeinerten *Hamming-Ähnlichkeit* [Hamming 1950] nach Formel 4.13 (vgl. [Mikulakova 2010]).

$$\bigwedge_{a_i \in b_i, a_j \in b_j} \bigvee_{a_{n_i} = a_{n_j}} sim_{cont}(b_i, b_j) := 1 - \frac{\sum \left(w_i \cdot \left| \frac{a_{w_i}}{m} - \frac{a_{w_j}}{m} \right| \right)}{n}$$

mit $1 \leq i \leq n$ und $1 \leq j \leq n$
mit n ist die Anzahl übereinstimmender Attribute, es gilt: $n := \left|A_{N_i} \cap A_{N_j}\right|$
mit $b_i \in B \wedge b_j \in B$
mit $m := \left|a_{w_i} - a_{w_j}\right|$
und w_i ist die Wichtung für das Attribut a_i

sim_{cont} Inhaltliches Ähnlichkeitsmaß (engl. Content Similarity) (4.13)

[7] Literale oder boolesche Werte werden zu diesem Zweck in einen für Formel 4.13 verwertbaren, numerischen Wert überführt.

Haben die Bauteilobjekte b_i und b_j keine gemeinsamen Attribute, ist die inhaltliche Ähnlichkeit $sim_{cont}(b_i, b_j) = 0$.

$$\left(\left|A_{N_i} \cap A_{N_j}\right| = 0\right) \Rightarrow sim_{cont} := 0 \tag{4.14}$$

Gesamtähnlichkeit Durch Addition der strukturellen und inhaltlichen Ähnlichkeit $sim_{struct}(b_i, b_j)$ und $sim_{cont}(b_i, b_j)$ ergibt sich die Gesamtähnlichkeit $sim_o(b_i, b_j)$ nach Formel 4.15 (vgl. [Mikulakova 2010]).

$$sim_o(b_i, b_j) := w_{struct} \cdot sim_{struct}(b_i, b_j) + w_{cont} \cdot sim_{cont}(b_i, b_j)$$

mit w_{struct} ist die Wichtung des strukturellen Ähnlichkeitsmaßes
mit w_{cont} ist die Wichtung des inhaltlichen Ähnlichkeitsmaßes
es gilt: $w_{struct} + w_{cont} = 1$

sim_o Gesamtähnlichkeitsmaß (engl. Overall Similarity) (4.15)

Die Werte w_{struct} und w_{cont} ermöglichen die Wichtung der strukturellen und inhaltlichen Ähnlichkeit bei der Zusammenführung beider Ähnlichkeitsmaße zum Gesamtähnlichkeitsmaß sim_o.

4.2.4 Abbildung von Vorgängen

Vorgänge werden ebenfalls als Objekte betrachtet und durch deren Eigenschaften beschrieben. Auch für deren Modellierung sind die sich aus dem Systementwurf nach Abbildung 4.1 abzuleitenden Anforderungen maßgebend.

Anforderungen

Ziel der Abbildung von Vorgängen ist deren formale Übertragbarkeit bezogen auf die Erstellung ähnlicher Bauteile - oder anders ausgedrückt:

Liefert ein Vorgang t als Ergebnis ein Bauteil b_1 und es existiert ein Bauteil b_2, das b_1 ähnlich ist, so kann b_2 durch den Vorgang t ebenfalls erzeugt werden. Gleichzeitig benötigt der Vorgang t die gleichen Voraussetzungen für dessen Ausführbarkeit (vgl. Definition 1 und Definition 2 auf Seite 29). Es gilt:

Definition 3 *Ein Vorgang hat ein Bauteil oder mehrere Bauteile in einem jeweils definierten Zustand als Voraussetzung.*

Definition 4 *Ein Vorgang liefert nach dessen Ausführung genau ein Bauteil in einem definierten Zustand als Ergebnis.*

Ein Vorgang hat demnach eine beliebige Anzahl von Voraussetzungen und genau ein Resultat. Voraussetzungen und Resultate werden durch Bauteilobjekte nach Kapitel 4.2.2 ausgedrückt. Weiterhin wird für die Abbildung eines Vorgangs ein Name sowie eine textuelle Beschreibung des Vorgangs benötigt.

Einschränkung: Wie bereits eingangs in Kapitel 2.5.2 erwähnt und in Abbildung 2.3 bildlich dargestellt, kann eine Folge von Vorgängen zu einem einzelnen Vorgang zusammengefasst werden. Dies ist notwendig, da bei entsprechend detaillierter Modellierung von Vorgängen die geforderte Abbildung von Vorgängen mit Vorbedingungen und Resultat entsprechend dem vorgestellten Lösungsansatz nicht eingehalten werden kann. Gründe hierfür sind beispielsweise Vorgänge, die nicht auf die Erstellung von Bauteilen zurückzuführen sind, wie das Einbringen von Beton in eine vorbereitete Schalung. Dies betrifft ebenso die Voraussetzungen solcher Vorgänge, die eher der Bereitstellung von Material oder Ressourcen entsprechen. Einem Vorgang kann jedoch ein beliebiger Substitutionsgraph zugeordnet werden, der im jeweils erforderlichen Detailgrad modelliert ist. Die Abbildung von Subgraphen für Vorgänge hat jedoch auf den zugrunde liegenden Ansatz dieser Arbeit keinen Einfluss. Aus diesem Grund und im Hinblick auf die Übersichtlichkeit wird auf die Beachtung von Substitutionsgraphen für Vorgänge im weiteren Verlauf der Arbeit verzichtet.

Formale Abbildung

Vorgang Formal werden Vorgänge als Menge von Objekten T (engl. Task) abgebildet. Jeder Vorgang $t \in T$ besteht aus dem Tripel (t_n, t_d, t_{guid}). Das Tripel besteht aus dem Namen des Vorgangs t_n, einer Beschreibung des Vorgangs t_d sowie dem eindeutigen, persistenten Identifikator t_{guid}.

$$
\begin{aligned}
&T := \{t_1, \ldots, t_i \mid t_i \text{ ist ein Vorgang}\} \text{ mit } i \in \mathbb{N} \\
&t_i := (t_n, t_d, t_{guid}) \\
\\
&t_n := (c_1, \ldots, c_i) \mid c_i \in \mathbb{T} \text{ - ist Name des Vorgangs} \\
&t_d := (c_1, \ldots, c_i) \mid c_i \in \mathbb{T} \text{ - ist Beschreibung des Vorgangs} \\
&t_{guid} \in ID \text{ - ist eindeutiger, persistenter Identifikator} \\
&\text{es gilt: } \bigwedge_{T_i, T_j} t_{guid_i} \neq t_{guid_j} \quad \text{für } i \neq j \text{ mit } i, j \in \mathbb{N}
\end{aligned}
\tag{4.16}
$$

Vorgangsgraph Die formale Abbildung von Vorgängen mit seinen Voraussetzungen P (engl. Prerequisites) und seinen Resultaten R (engl. Results) erfolgt auf Basis eines bipartiten Graphen[8]. Für die Abbildung eines Vorgangs mit seinen Voraussetzungen und Resultaten wird der Begriff Vorgangsgraph TG (engl. Task Graph) eingeführt. Für die Abbildung des Graphen TG ergeben sich vier Objektmengen. Es handelt sich dabei

[8] siehe Kapitel 3.1 und [Pahl u. Damrath 2000]

um die Menge der Vorgänge T (engl. Task), die Menge der Bauteile B (engl. Building Element) und die zwischen diesen Mengen existierenden Relationen P und R.

$$TG := (T, B; P, R)$$

$$\begin{aligned}&\text{mit } R \subseteq T \times B;\ P \subseteq B \times T\\ &\text{und } PR^{+} \sqcap PR^{+T} = 0 \text{ für } PR := P \cup R\\ &\text{und } |T| = 1 \wedge |R| = 1\end{aligned}$$

TG	Vorgangsgraph
T	Menge der Vorgänge (engl. Task)
B	Menge der Bauteilbeschreibungen (engl. Building Element)
P	Voraussetzungen des Vorgangs (engl. Prerequisites)
R	Resultate des Vorgangs (engl. Results) (4.17)

Die Voraussetzungen P eines Vorgangs werden durch die Abbildung der Menge der Bauteile B auf die Menge der Vorgänge T gebildet und entsprechen somit den Kanten $p(b,t) \in P$. Die Resultate R eines Vorgangs werden durch die Abbildung der Menge der Vorgänge T auf die Menge der Bauteile B definiert und entsprechen demnach den Kanten $r(t,b) \in R$. Dabei ist die Anzahl der Vorbedingungen nicht begrenzt. Es wird jedoch gefordert, dass ein Vorgang nur ein Resultat $r(t,b)$ liefert. Die Menge der Vorgänge T eines *Vorgangsgraphen* TG ist ebenfalls auf einen Vorgang beschränkt. Weiterhin wird gefordert, dass ein *Bedingungsgraph* azyklisch[9] ist. Oder anders ausgedrückt, ein Bauteil in einem definierten Zustand, welches bereits Vorbedingung eines Vorgangs ist, kann nicht gleichzeitig ein Ergebnis desselben Vorgangs sein.
Abbildung 4.4 zeigt das Modell als graphische Darstellung.

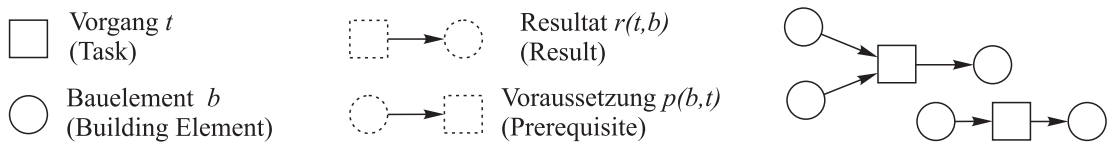


Abbildung 4.4: Beispiel für Vorgänge mit Voraussetzungen und Resultaten (*Vorgangsgraphen*)

Teilmengen Innerhalb der Domäne existieren, analog zu den Mengen von Bauteilen, Teilmengen von Vorgangsgraphen. Es handelt sich um die Menge der Vorgangsgraphen des betrachteten BIM TG_{BIM}, die Menge der Vorgangsgraphen der Datenbasis TG_{DB} sowie die Menge von Vorgangsgraphen TG_{SU_i}, die einer Struktureinheit des BIM zugeordnet wurden.

[9] vgl. [Pahl u. Damrath 2000] Ein bipartiter Graph TG ist azyklisch, wenn die transitive Hülle PR^{+} mit $PR := P \cup R$ asymmetrisch ist. Existiert ein nichtleerer Weg von x nach y, so existiert kein nichtleerer Weg von y nach x, da sonst die Verkettung beider Wege einen Zyklus liefern würde. TG ist azyklisch $\Leftrightarrow PR^{+} \sqcap PR^{+T} = 0$

$$TG_{BIM} := \{tg_1, \ldots, tg_i \mid tg_i \text{ ist Vorgangsgraph des BIM}\} \text{ mit } i \in \mathbb{N}$$
$$TG_{SU_j} := \{tg_1, \ldots, tg_k \mid tg_k \text{ ist Vorgangsgraph von } SU_j\} \text{ mit } j, k \in \mathbb{N}$$
$$TG_{DB} := \{tg_1, \ldots, tg_n \mid tg_n \text{ ist Vorgangsgraph der Datenbasis}\} \text{ mit } n \in \mathbb{N}$$

$$\text{und } TG_{SU_j} \subset TG_{BIM} \subset TG_{DB} \tag{4.18}$$

Beispiel: Das nachfolgende Beispiel zeigt die formale Abbildung zweier möglicher Vorgänge für das Beispiel nach Abbildung 4.3. Vorgang t_1 erstellt Fundamente mit den Eigenschaften nach Bauteilbeschreibung b_2. Die Angabe von *null* in P_1 bedeutet, dass der Vorgang t_1 keine notwendige Voraussetzung hat. Vorgang t_2 erstellt Wände mit den Eigenschaften nach Bauteilbeschreibung b_1 und hat als notwendige Voraussetzung ein Bauteil mit den Eigenschaften des Bauteils b_2.

$TG_1 := T_1, B_1, P_1, R_1$ $\quad$ $TG_2 := T_2, B_2, P_2, R_2$

$T_1 := \{t_1\}$ $\quad$ $B_1 := (b_2)$ $\quad$ Siehe Beispiel Seite 66
$T_2 := \{t_2\}$ $\quad$ $B_2 := (b_1, b_2)$ $\quad$ Siehe Beispiel Seite 66

$t_1 := (t_{n_1}, t_{d_1}, t_{guid_1})$
$t_{n_1} :=$ "*Fundament erstellen*" $\quad$ $t_{d_1} :=$ "*Vorgang erstellt ein Fundament . . .*"
t_{guid_1}" := "*ghjuadGbMIsekb*55*sJHiz*2"

$P_1 := \{(null, t_1)\}$ $\quad$ $R_1 := \{(t_1, b_2)\}$

$t_2 := (t_{n_2}, t_{d_2}, t_{guid_2})$
$t_{n_2} :=$ "*Wand erstellen*" $\quad$ $t_{d_2} :=$ "*Vorgang erstellt eine Ziegelwand . . .*"
t_{guid_2}" :=" *oplzdhGbMIsekb*29*sJJe*38"

$P_2 := \{(b_2, t_2)\}$ $\quad$ $R_2 := \{(t_2, b_1)\}$

4.2.5 Abbildung der Datenbasis

Als Datenbasis wird die Abbildung aller im System definierten *Vorgangsgraphen* TG_{DB} verstanden. Wird ein neuer Vorgang nach Kapitel 4.2.4 definiert, wird er in Form eines *Vorgangsgraphen* TG_i in die Datenbasis TG_{DB} eingetragen und steht ab diesem Zeitpunkt für alle nachfolgenden Generierungsvorgänge nach Kapitel 4.3.2 zur Verfügung. Die Menge der *Vorgangsgraphen* TG_{DB} wächst mit jedem neu definierten Vorgang. Die Datenbasis stellt somit eine „lernende“ Wissensbasis dar. Sie repräsentiert alle im System enthaltenen Vorgänge in Form von beschreibenden Datensätzen für die Erstellung von Bauelementen und hält diese für den Generierungsprozess vor. Es wird dabei nicht

gefordert, dass genau ein Vorgang für die Herstellung eines konkreten Bauelements existiert. Für die Herstellung desselben Bauteils kann eine beliebige Anzahl weiterer Vorgänge mit demselben Resultat definiert sein.

Formale Abbildung

Die Datenbasis wird formal durch die Gesamtmenge aller in der Datenbasis enthaltenen *Vorgangsgraphen* (siehe Formel 4.18) abgebildet und entspricht der Menge TG_{DB}.

4.2.6 Abbildung der Grobstrukturierung

Die Notwendigkeit der Abbildung einer Grobstrukturierung des Bauwerks leitet sich aus dem Systementwurf nach Kapitel 4.1 Punkt 1 ab und muss die nachfolgenden Anforderungen erfüllen.

Anforderungen

Eine Struktureinheit beschreibt einen Teil des geplanten Bauwerks oder Baumaßnahme und besteht demnach aus einer endlichen Anzahl von Bauteilen. Die Menge der Bauteile einer Struktureinheit muss zu diesem Zweck aus der Menge der Bauteile des BIM extrahiert werden können.

Weiterhin ist die Modellierung von gerichteten Abhängigkeiten zwischen Struktureinheiten notwendig, um deren vom Anwender zu definierende Bearbeitungsreihenfolge im Bauablauf abzubilden.

Formale Abbildung

Struktureinheit Formal werden Struktureinheiten als die Menge von Objekten SU (engl. Structure Units) abgebildet. Jede Struktureinheit $su \in SU$ besteht aus einem 6-Tupel[10] (su_n, su_r, su_{guid}, su_b, su_{tg}, su_s). Im Einzelnen bezeichnet su_n den Namen der Struktureinheit, su_r die Bildungsvorschrift für deren Definition (siehe Kapitel 4.3.1), su_{guid} einen eindeutigen, persistenten Identifikator, su_b die Menge der auf Basis der Bildungsvorschrift su_r aus dem BIM extrahierten Bauelemente $b \in B_{BIM}$, su_{tg} die Menge der dieser Struktureinheit zugeordneten Vorgangsgraphen (siehe Kapitel 4.3.2) und dem generierten Bauablauf für dieser Struktureinheit su_s (siehe Abschnitt „Verknüpfungsphase“ auf Seite 89).

[10] geordnete Menge von sechs Elementen, siehe [Scheid 2000]

$$
\begin{aligned}
&SU := \{su_1, \ldots, su_i \mid su_i \text{ ist eine Struktureinheit des BIM}\} \text{ mit } i \in \mathbb{N} \\
&su_i := (su_n, su_r, su_{guid}, su_b, su_{tg}, su_s) \\
&\\
&su_n := (c_1, \ldots, c_i) \mid c_i \in \mathbb{T} \text{ - ist Name der Struktureinheit} \\
&su_r := (c_1, \ldots, c_i) \mid c_i \in \mathbb{T} \text{ - ist Bildungsvorschrift der Struktureinheit (engl. Rule)} \\
&su_{guid} \in ID \text{ - ist eindeutiger, persistenter Identifikator} \\
&\\
&\text{es gilt: } \bigwedge_{SU_i, SU_j} su_{guid_i} \neq su_{guid_j} \quad \text{für } i \neq j \text{ mit } i, j \in \mathbb{N} \\
&su_b := \{b_1, \ldots b_k \mid b_k \in B_{SU_i}\} \text{ für } k \in \mathbb{N} \\
&su_{tg} := \{tg_1, \ldots, tg_i) \mid tg_i \in TG_{DB} \text{ - ist ein Vorgangsgraph} \\
&su_s := S(T, B, P, R)
\end{aligned}
\tag{4.19}
$$

Strukturgraph Die gerichteten Abhängigkeiten zwischen den Struktureinheiten $su_i \in SU$ erfolgt mittels eines schlichten, gerichteten Graphen[11] SG (engl. Structure Graph). Es wird gefordert, dass der Graph SG azyklisch ist, so dass die Vorgänge zum Erstellen von Bauelementen einer Struktureinheit im angestrebten Bauablauf nur einmal ausgeführt werden können.

$$
\begin{aligned}
&SG := (SU, O) \text{ mit } O \subseteq SU \times SU \\
&\text{und } O^+ \sqcap O^{+T} = 0 \\
&\\
&SG \quad \text{Strukturgraph (engl. Structure Graph)} \\
&O \quad \text{Reihenfolge der Struktureinheiten (engl. Order)}
\end{aligned}
\tag{4.20}
$$

4.2.7 Abbildung des Bauablaufs

Ziel dieser Arbeit ist das teilautomatisierte Generieren von Bauabläufen. Das Ergebnis des angestrebten Automatisierungsprozesses ist ein Bauablaufplan. Für dessen Abbildung bestehen folgende Anforderungen:

[11] siehe [Pahl u. Damrath 2000], Kapitel 8.3.2

Anforderungen

Das gesuchte Modell muss aufgrund der im angestrebten Bauablauf enthaltenen Varianten für die Abbildung von Entscheidungen und parallelen Abläufen geeignet sein. In der Literatur existieren verschiedene Modelle, um Entscheidungssituationen und Nebenläufigkeit abzubilden. Als besonders geeignet haben sich die auf der Basis von Petri-Netzen weiterentwickelten Workflowgraphen etabliert [van der Aalst 1998; König 2004; Klinger 2003]. Ein Workflowgraph ist ein bipartiter Graph, durch dessen Abbildung zweier Knotenmengen parallele und alternative Vorgänge formal definiert werden können. Für Workflowgraphen gelten gegenüber bipartiten Graphen zusätzliche Bedingungen hinsichtlich ihrer strukturellen Integrität. Erläuterungen, verwendete Methoden und Algorithmen wurden in Kapitel 3.4 bezogen auf den angestrebten Lösungsansatz dargestellt.

Formale Abbildung

Der Bauablauf S entspricht infolge des Systementwurfs der Menge aller identifizierten, gültigen Vorgangsgraphen der Datenbasis TG_{DB}, die den Bauteilbeschreibungen B_{BIM} des BIM zugeordnet werden konnten (vgl. Kapitel 4.3.2), sowie zusätzlichen Scheinknoten und -kanten infolge der Konfliktlösungs- (vgl. Kapitel 4.3.2) und Kopplungsphase (vgl. Kapitel 4.3.3). Es wird gefordert, dass der Bauablauf S azyklisch ist.

$$S := \{T_S, B_S, P_S, R_S\} \mid S \text{ ist ein Workflowgraph}$$

$$\text{mit } PR^{+} \sqcap PR^{+T} = 0 \text{ für } PR := P_S \cup R_S$$

S	Workflowgraph des Bauablaufs (engl. Schedule)	
T_S	Menge der Vorgänge des Bauablaufs (engl. Task)	
B_S	Menge der Bauteilbeschreibungen des Bauablaufs (engl. Building Element)	
P_S	Voraussetzungen der Vorgänge des Bauablaufs (engl. Prerequisites)	
R_S	Resultate der Vorgänge des Bauablaufs (engl. Results)	(4.21)

4.3 Methoden für die Generierung des Bauablaufs

Die nachfolgenden Unterkapitel behandeln Methoden und Algorithmen für den Gesamtprozess der Generierung. Dieser wird in der Reihenfolge der drei Phasen des Systementwurfs nach Kapitel 4.1 diskutiert (siehe Abbildung 4.1 und 4.5).

Abbildung 4.5: Phasen der Generierung

4.3.1 Strukturierungsphase

Innerhalb der Strukturierungsphase erfolgt sowohl die Einteilung des durch das BIM gegebenen Bauwerks in Struktureinheiten bzw. Bauabschnitte SU, für die ein Teilablauf erstellt werden soll, als auch die Definition von deren Reihenfolge in Form eines Strukturgraphen SG im Bauablauf durch den Anwender. Ziel ist die Extraktion und Zuordnung der einer Struktureinheit zugehörigen Bauteile. Die Bildung dieser Einheiten ist dabei an die Vorgehensweise der Grobterminplanung beim Erstellen eines Bauablaufplans angelehnt.

In [Tulke 2010] erfolgt eine Betrachtung der herkömmlichen Vorgehensweise für die Gliederung in Bauabschnitte und deren Überführung in eine modellbasierte Arbeitsweise auf Basis eines BIM. Demnach erfolgt eine erste Einteilung auf Basis von Gewerken im Sinne einer groben Teilung der auszuführenden Arbeiten für den Roh- und Ausbau, wobei Ausbauarbeiten in der Regel raumbezogen definiert werden. Sowohl Rauminformationen als auch die Referenzierung darin enthaltener Objekte ist in der Regel Bestandteil eines BIM und somit direkt verfügbar.

Die Definition von Bauabschnitten bezogen auf auszuführende Rohbauarbeiten wird hingegen zumeist auf Basis der groben Bauwerksstruktur festgelegt und im Anschluss weiter untergliedert. Diese Untergliederung erfolgt in der Regel basierend auf einer zweidimensionalen Arbeitsweise innerhalb von Zeichnungen. Zu diesem Zweck werden vom Architekten Achsen und Arbeitszonen definiert. Sie werden dazu in die dem Bauwerk zugrunde liegenden Zeichnung eingepflegt, farbig markiert und mit einer eindeutigen Bezeichnung versehen. Dies dient der Referenzierung der Bauabschnitte innerhalb des Beschreibungstextes für die auszuführenden Arbeiten - beispielsweise für die Definition von Betonierabschnitten zwischen zwei benannten Achsen oder der Herstellung von Bauteilen oder Ausbaustufen innerhalb einer Arbeitszone. Bei der Übertragung dieser Vorgehensweise muss festgestellt werden, dass die in einem BIM enthaltenen Daten einer solchen Einteilung nicht zwangsläufig entsprechen. Beispielsweise liegt eine Geschossdecke in dem der CAD-Applikation entstammenden BIM gemeinhin als ein über das komplette Geschoss durchgängiges Objekt vor.

In [Tulke 2010] wird unter anderem eine BIM-basierte Methodik für die Ermittlung von Bauteilen bezogen auf Arbeitszonen und Achseinteilungen vorgestellt, die der Zielsetzung der Strukturierungsphase in vollem Umfang entspricht. Das dort entwickelte System ermöglicht nicht nur die Selektion von Bauteilen über die Definition von Achsen und Arbeitszonen innerhalb des BIM, sondern darüber hinaus eine attributbasierte Auswahl von Bauteilen. Die Teilung von Bauteilen anhand definierter Arbeitszonen und Achsen wird vom System ebenfalls unterstützt.

Das System lässt sich über eine für diesen Zweck entwickelte Sprache ansprechen. Das Ergebnis einer solchen Anfrage liefert bei einer darauf ausgelegten Anfrage die Menge aller einer solchen Anfrage entsprechenden Bauteilidentifikatoren (vgl. Abschnitt „Anforderungen“ auf Seite 60) und ermöglicht auf diese Weise das Selektieren und Überführen der für den Ansatz dieser Arbeit notwendigen Bauteilinformationen aus dem BIM in das Modell nach Kapitel 4.2.

Auf die volle Darstellung des Sprachumfangs wird an dieser Stelle verzichtet. Es sind prinzipiell alle bauteilbezogenen Attribut-, Zonen- und Achsenabfragen in beliebiger

Kombination realisierbar. Es gilt:

$$su_{b_i} := select(su_{r_i})$$

mit $select(su_{r_i})$ ist eine Systemanfrage nach [Tulke 2010]
und su_{b_i} ist die Menge der nach Regel su_{r_i} selektierten Bauteile (4.22)

Zur Veranschaulichung der Systemanfragen $select(su_r)$ sollen die nachfolgenden zwei Beispiele dienen:

Systemanfrage - Beispiel 1: Die nachfolgend formulierte Systemanfrage selektiert die Menge aller Bauteilidentifikatoren *GUID* für die Menge der Bauteile im BIM (B_{BIM}) deren Attribut *BUILDING_STOREY* den Wert "2.*OG*"hat:

$select(\{B_{BIM} : BUILDING\_STOREY =\text{"}2.OG\text{"} -> GUID\})$

Abbildung 4.6 zeigt das Ergebnis der erfolgten Anfrage. Die durch die Anfrage selektierten Bauteile sind gelb dargestellt.

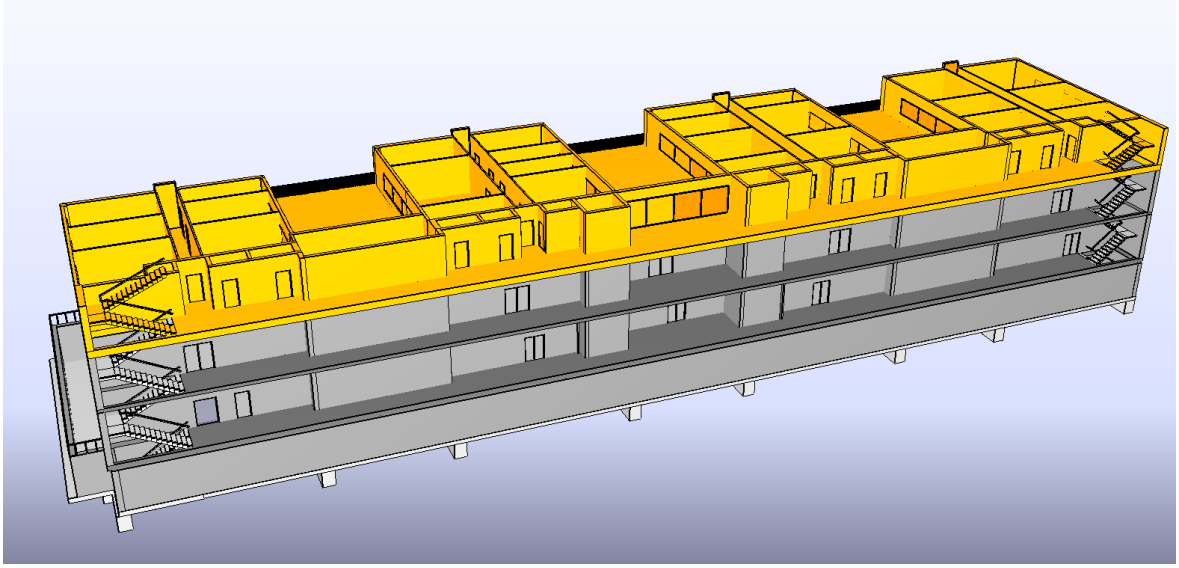


Abbildung 4.6: Selektion aller Bauteile im 2.OG nach Beispiel 1

Systemanfrage - Beispiel 2: Durch die Schachtelung von Abfragen kann mittels unten stehender Anweisung die Menge aller Bauteilidentifikatoren *GUID* für die Menge der Bauteile im BIM B_{BIM} selektiert werden, deren Attribut *MATERIAL* den Wert "*Stahlbeton*"hat und sich innerhalb der geometrisch definierten *ZoneA* befinden:

$select(\{inZoneRelation(\{B_{BIM} : MATERIAL =\text{"}Stahlbeton\text{"} -> \text{GUID}\}, \{\text{"}ZoneA\text{"}\}) : -> GUID\})$

Abbildung 4.7 zeigt das Ergebnis nach Ausführung der Anfrage nach Beispiel 2. Die innerhalb der Anfrage verwendete *ZoneA* wurde vom Anwender definiert und ist grün dargestellt. Die gelb markierten Bauteile befinden sich komplett innerhalb von *ZoneA* und weisen eine Materialzuordnung entsprechend der Anfrage auf.

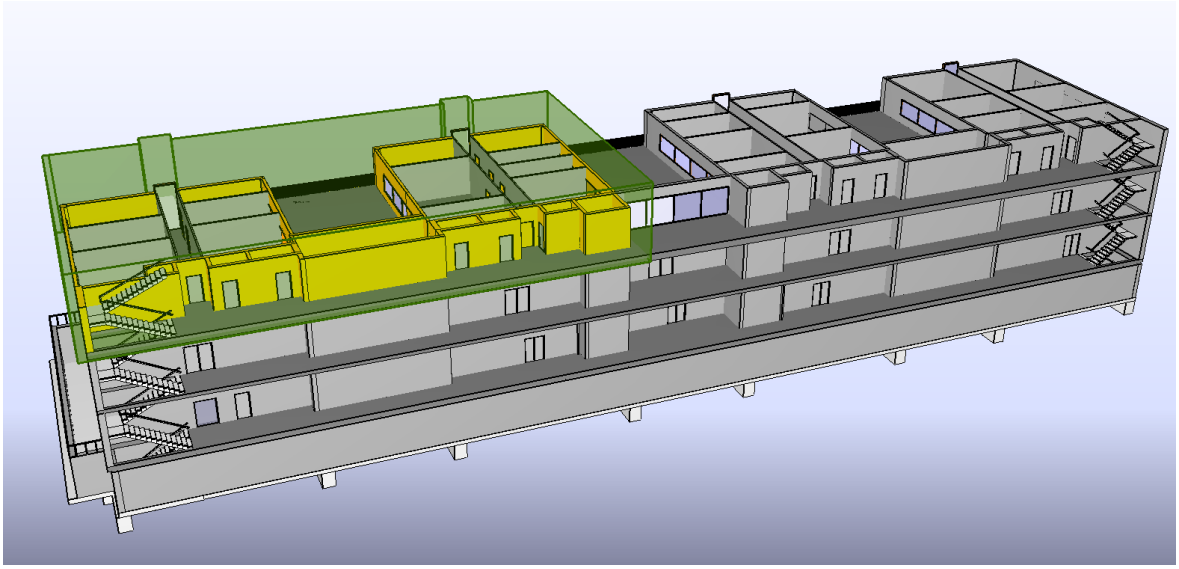


Abbildung 4.7: Selektion aller Bauteile in *ZoneA* mit dem Material "*Stahlbeton*" nach Beispiel 2

Nachfolgendes Beispiel verdeutlicht die Definition eines Strukturgraphen:

Strukturgraph - Beispiel: Das nachfolgende Beispiel zeigt die Definition von zwei Struktureinheiten su_1 und su_2. Struktureinheit su_1 bildet alle Bauelemente des Geschosses mit der Bezeichnung „Gründung" ab, su_2 alle Bauelemente des Geschosses mit der Bezeichnung „EG". Dies geschieht unter der Annahme, dass sich b_2 als einziges Bauelement im Geschoss „Gründung" bzw. b_1 als einziges Bauelement im Geschoss „EG" befindet und die Arbeiten Geschoss „Gründung" vor den Arbeiten am Geschoss „EG" stattfinden sollen. Die bauteilbeschreibenden Objekte b_1 und b_2 ergeben sich aus der jeweiligen Systemanfrage su_{r_1} und su_{r_2} nach [Tulke 2010] und werden nach deren Überführung in das Modell nach Kapitel 4.2.2 in die Menge su_{b_1} bzw. su_{b_2} eingetragen.

$SU := \{su_1, su_2\}$

$su_1 := (su_{n_1}, su_{r_1}, su_{guid_1}, su_{b_1})$

su_{n_1} := "Gründung"
su_{r_1} := "select({B_{BIM}:BUILDING_STOREY="Gründung"->GUID})"
su_{guid_1}" := "$gr2gc1GbMIseknaI1JHiz2$"
$su_{b_1} := \{b_2\}$ Siehe Beispiel auf Seite 66
$su_{tg_1} := \varnothing$

$su_2 := (su_{n_2}, su_{r_2}, su_{guid_2}, su_{b_2})$

su_{n_2} := "Erdgeschoss"
su_{r_2} := "select({B_{BIM}:BUILDING_STOREY="EG"->GUID})"
su_{guid_2}" := "$ghjuadidualCJHiz2$"
$su_{b_2} := \{b_1\}$ Siehe Beispiel auf Seite 66
$su_{tg_2} := \varnothing$

$SG := (SU, O)$

$O := \{(su_1, su_2)\}$

4.3.2 Generierungsphase

Soll für ein Bauprojekt auf Basis eines BIM eine Bauablaufplanung erstellt werden, wird für jedes laut Planung herzustellende Bauteil des Bauwerks in der Datenbasis nach Vorgängen gesucht, die ein jeweils ähnliches Bauteil als Ergebnis haben. Als notwendige Ausgangsinformation wird die Menge von Bauteilen verstanden, für deren Herstellung ein Workflowgraph generiert werden soll. Es handelt sich somit um alle Bauteile jeder definierten Struktureinheit des geplanten Bauwerks bzw. der geplanten Baumaßnahme. Der Gesamtprozess der Generierungsphase wird demnach für jede Struktureinheit ausgeführt und kann in die drei Phasen gegliedert werden. Es handelt sich im Einzelnen um die *Zuordnungsphase*, die *Verknüpfungsphase* und die *Konfliktlösungsphase*. Abbildung 4.8 zeigt die Gliederung der Generierungsphase als Flussdiagramm.

Zuordnungsphase

Die Zuordnungsphase ist in zwei automatisierte und gegebenenfalls zwei weitere manuelle Arbeitsschritte gegliedert, die notwendig werden, falls die automatisierten Arbeitsschritte nicht zu einer erfolgreichen Zuordnung führen:

1. Im ersten Schritt wird für jedes Bauteil einer Struktureinheit innerhalb der Datenbasis nach einem Vorgang gesucht, der ein ähnliches Bauteil als Ergebnis hat. Wird ein solcher Vorgang gefunden, so wird der jeweilige Ergebnisknoten des Bedingungsgraphen mit dem entsprechenden Bauteil verknüpft. Eine Mehrfachverknüpfung ein und desselben Bauteils mit verschiedenen Vorgängen ist hierbei möglich. Konnte mehr als ein passender Vorgang für ein Bauteil ermittelt werden, existieren verschiedene Vorgänge, die zur Erstellung eines identischen Bauteils führen.

   Konnte nicht für alle Bauteile der Struktureinheit ein passender Vorgang gefunden werden, ist die manuelle Definition eines solchen Vorgangs durch den Anwender notwendig. Der durch den Anwender definierte Vorgang wird in die Datenbasis eingefügt und steht ab diesem Zeitpunkt für die Generierungsphase zur Verfügung. Nach einer solchen Neudefinition eines Vorgangs muss der Versuch der Zuordnung für alle zu betrachtenden Bauteile wiederholt werden. Dies ist notwendig, da der oder die neu erzeugten Vorgänge Auswirkungen auf andere, noch nicht zugeordnete oder auch bereits zugeordnete Bauteile haben kann (vgl. Zuordnung von Vorgängen, Seite 82).

2. Der zweite Schritt betrifft die Zuordnung notwendiger Voraussetzungen. Dies betrifft alle Vorgänge, die im ersten Schritt einem Bauteil zugeordnet wurden. Es wird versucht, den Voraussetzungsknoten eines jeden ermittelten Vorgangs ein entsprechendes Bauteil der Ausgangsmenge zuzuordnen. Dafür wird in der Ausgangsmenge nach Bauteilen gesucht, die der Bauteilbeschreibung der jeweilig betrachteten Voraussetzung ähnlich ist. Wird ein solches Bauteil erkannt, wird dessen geometrische Nähe zum bestimmten Bauteil des Ergebnisknotens als weiteres Kriterium für eine Identifikation als Voraussetzung eines Vorgangs herangezogen.

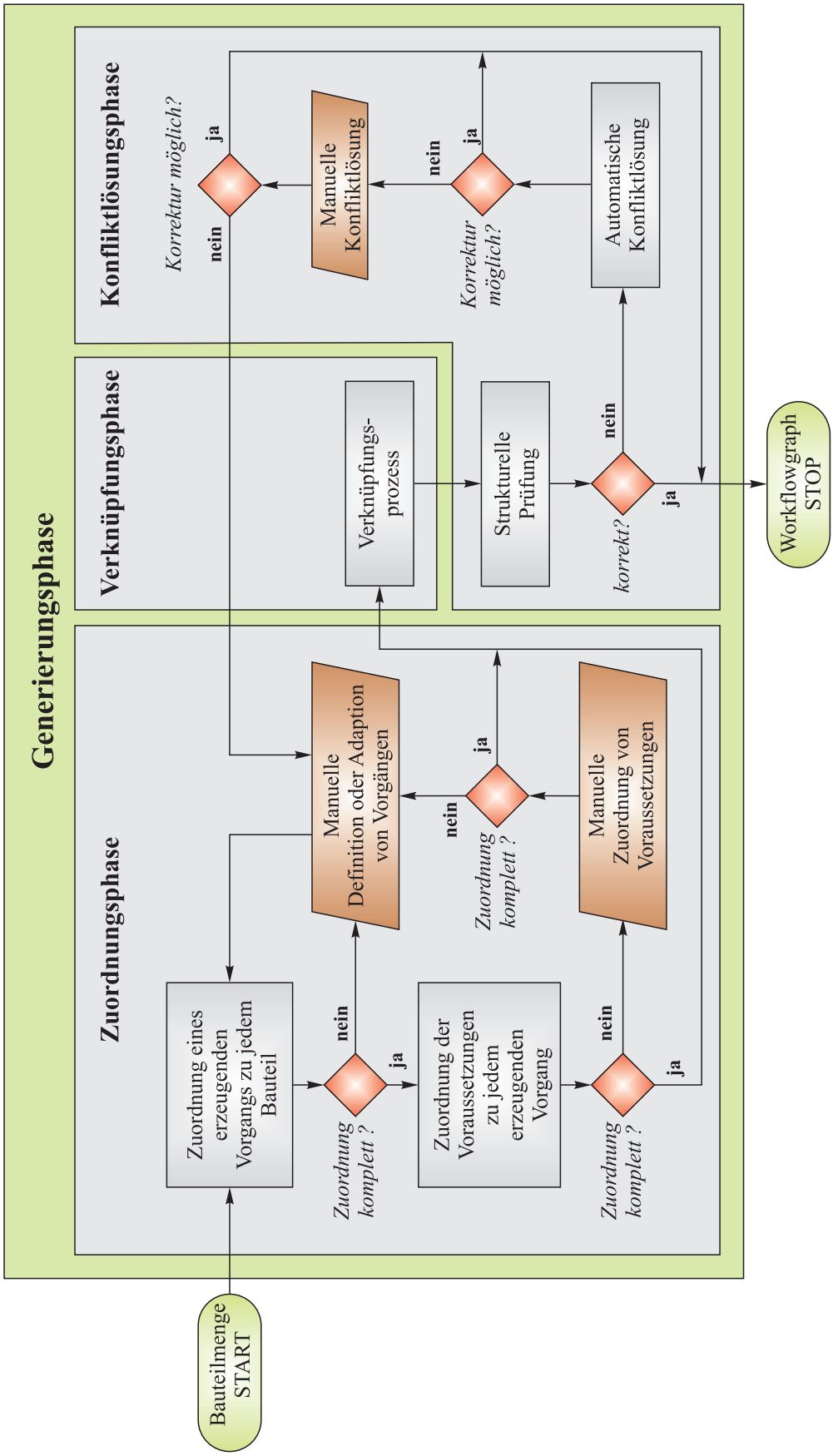


Abbildung 4.8: Flussdiagramm des Generierungsprozesses

Bei erfolgreicher Bestimmung eines solchen Bauteils erfolgt die Verknüpfung mit dem entsprechenden Voraussetzungsknoten des Bedingungsgraphen.

Konnten nicht alle Voraussetzungen der im ersten Schritt bestimmten Vorgänge automatisch zugeordnet werden, erfolgt zunächst die manuelle Zuordnung dieser Voraussetzungen durch den Anwender. Ist eine manuelle Zuordnung nicht möglich, gibt es keinen Vorgang, der das entsprechende Bauteil erzeugen kann. Es muss demnach ein neuer Vorgang durch den Anwender definiert werden. Nach einer solchen Definition muss der erste Schritt der Zuordnungsphase erneut ausgeführt werden, da die Neudefinition von Vorgängen auch an dieser Stelle im Algorithmus Auswirkungen auf andere Bauteile der Ausgangsmenge haben kann.

Die automatisierten Schritte der Zuordnungsphase werden im Anschluss detailliert erläutert. Für eine besseres Verständnis der Algorithmen und Methodik wird nachfolgendes Beispiel zugrunde gelegt:

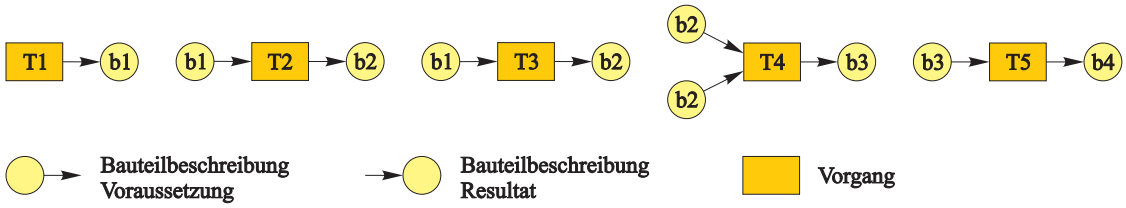


Abbildung 4.9: Definierte Vorgangsgraphen der Datenbasis TG_{DB}

Für das Beispiel wird vorausgesetzt, dass die nachfolgend näher erläuterten Bauteilbeschreibungen $b1$ – $b4$ entsprechend ihrer Ausprägung eine sehr hohe Ähnlichkeit mit den in Abbildung 4.10 bezeichneten Bauteilen aufweisen. Die in Abbildung 4.9 dargestellten Vorgangsgraphen haben hierbei folgende Bedeutung:

T1 Vorgang mit dem Resultat einer Decke gegeben durch die Bauteilbeschreibung **b1** und keiner Voraussetzung

T2 Vorgang mit dem Resultat einer Stahlstütze gegeben durch die Bauteilbeschreibung **b2** und die Voraussetzung einer Decke, beschrieben durch **b1**

T3 Vorgang mit dem Resultat einer Stahlstütze gegeben durch die Bauteilbeschreibung **b2** und die Voraussetzung einer Decke, beschrieben durch **b1**

T4 Vorgang mit dem Resultat einer vorgehängten Fassadenwand gegeben durch die Bauteilbeschreibung **b3** und zwei Stahlstützen als Voraussetzungen, beschrieben durch **b2**

T5 Vorgang mit dem Resultat einer Fensterfront gegeben durch die Bauteilbeschreibung **b4** und die Voraussetzung einer Fassadenwand vom Typ **b3**

Anmerkung: Die Vorgänge **T2** und **T3** sind nicht identisch. Es handelt sich um zwei unterschiedliche Verfahren, die eine Stahlstütze mit der selben Bauteilbeschreibung **b** als Resultat und der selben Bauteilbeschreibung **a** einer Decke als Voraussetzung aufweisen.

Für die Bezeichnung der Bauteile des BIM wird die in Abbildung 4.10 dargestellte Nummerierung verwendet. Die mit Nummer 1 bis 9 bezeichneten Bauteile stellen die Bauteilmenge su_b der jeweils betrachteten Struktureinheit su dar.

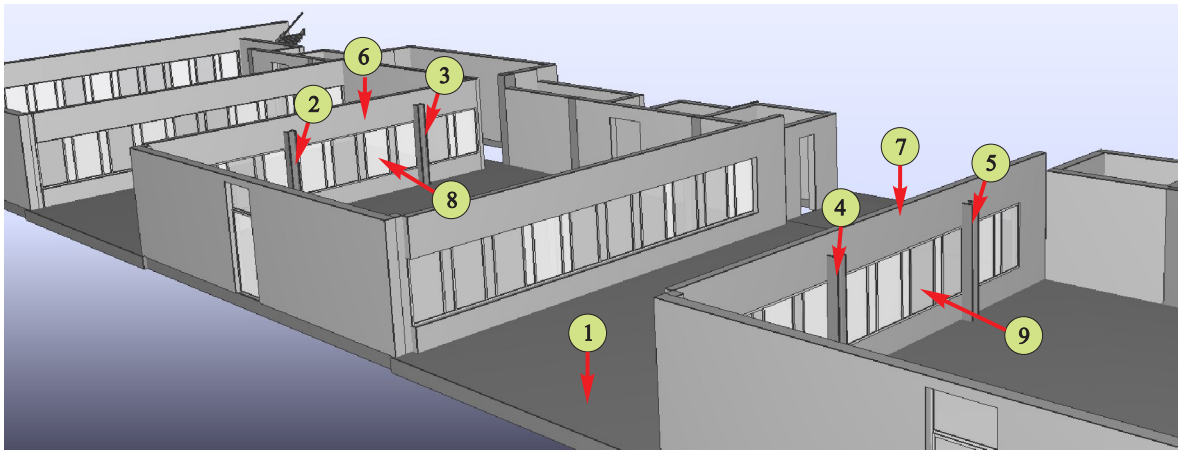


Abbildung 4.10: Bauteilbezeichnungen von $b_i \in su_b$ der als Beispiel verwendeten Struktureinheit

Schritt 1 - Zuordnung von Vorgängen zu Bauteilen Die Zuordnung eines Vorgangs $t \in T_{DB}$ zu einem Bauteilobjekt $b_i \in su_b$ einer Struktureinheit su erfolgt auf Basis des Vergleichs von Bauteilbeschreibungen der Datenbasis B_{DB} mit den Bauteilobjekten su_b einer Struktureinheit. Algorithmus 16 zeigt die notwendigen Arbeitsschritte als Pseudocode.

Beschreibung Algorithmus 16: Als Eingangswerte dienen die zu betrachtende Struktureinheit su, der Ähnlichkeitsgrenzwert lim_{sim} sowie die Menge der Vorgangsgraphen der Datenbasis TG_{DB}.
Es werden alle Bauteilobjekte $currentBE \in su_b$ der betrachteten Struktureinheit su mit allen Bauteilbeschreibungen $b_{tg} \in B_{DB}$ der Datenbasis verglichen, die ein Resultat eines Vorgangs $r(t_k, b_{tg}) \in R$ darstellen. Sind das Bauteilobjekt $currentBE$ und die Bauteilbeschreibung b_{tg} einander ähnlich, wird der Vorgang t_{tg} geklont (*deep cloning*)[12] und das konkrete Bauteilobjekt $currentBE$ als Resultat zugeordnet. Der Ähnlichkeitsgrenzwert $lim_{sim} \mid 0 \leq lim_{sim} \geq 1$ kennzeichnet den unteren Grenzwert für eine solche Zuordnung. Eine Zuordnung eines Vorgangs erfolgt nur, wenn die Bedingung

[12]Unter klonen (engl. clone) wird in der Datenverarbeitung das Duplizieren eines Objekts verstanden. Man unterscheidet zwischen *deep cloning*, wobei der Klon des Objekts ein eigenständiges Objekt mit den gleichen, jedoch ebenfalls geklonten Eigenschaften darstellt, und einer *shallow copy*, bei welcher der Klon ein eigenständiges Objekt ist, seine Eigenschaften jedoch identisch sind, also auf die Eigenschaftsobjekte des zu duplizierenden Objektes referenzieren.

Algorithmus 16: determineResults(su, lim_{sim}, TG_{DB})
Zuordnung von Vorgängen zu Bauteilen

input : $su(su_n, su_r, su_{guid}, su_b, su_{tg}, su_s)$, lim_{sim}, TG_{DB}
output : Menge der Vorgangsgraphen $su_{tg} \in su$

```
1  su_tg := ∅;
2  foreach currentBE ∈ su_b do
3      foreach currentTG(T, B, P, R) ∈ TG_DB do
4          foreach r(t_tg, b_tg) ∈ R do
5              if sim_o(currentBE_su, b_tg) ≥ lim_sim then
6                  T_temp := {deepClone (t_tg)};
7                  B_temp := B \ {b_tg} ∪ {currentBE_su};
8                  P_temp := P;
9                  R_temp := {(t_tg, currentBE_su)};
10                 tempTG := (T_temp, B_temp, P_temp, R_temp);
11                 su_tg := su_tg ∪ {tempTG}
12             end
13         end
14     end
15 end
```

$sim_0(currentBE, b_{tg}) \geq lim_{sim}$ erfüllt ist. Der Vorgang t_{tg} stellt in diesem Zusammenhang einen möglichen Vorgang für die Erstellung des Bauteilobjektes *currentBE* dar und wird in die Menge der Vorgangsgraphen su_{tg} der Struktureinheit su eingetragen. Eine manuelle Definition eines neuen Vorgangs bzw. Vorgangsgraphen ist notwendig, falls nicht für alle Bauteile der Struktureinheit su ein Vorgang bestimmt werden konnte. Dies ist der Fall, wenn der Betrag der Differenz aller Bauteile einer Struktureinheit su_b und der Menge der Bauteile aller Strukturgraphen su_{tg}, die Resultat eines Vorgangs sind, größer 0 ist.

$$su_{r_{left}} := su_b \setminus \{b_i \in B_{TG} \mid r(t, b_i) \in R_{TG}\}$$

$su_{r_{left}}$ ist die Menge der Bauteile der Struktureinheit, für die kein Vorgang identifiziert werden konnte

$$\text{mit } B_{TG} := \left\{ \bigcup_{j=1}^{|su_{tg}|} B_j \mid TG_j(T_j, B_j, P_j, R_j) \in su_{tg} \right\}$$
$$\text{und } R_{TG} := \left\{ \bigcup_{j=1}^{|su_{tg}|} R_j \mid TG_j(T_j, B_j, P_j, R_j) \in su_{tg} \right\} \tag{4.23}$$

Wurde ein neuer Vorgangsgraph tg_{new} definiert und in die Datenbasis TG_{DB} eingefügt, ist Algorithmus 16 erneut auszuführen.

Beispiel: Abbildung 4.11 zeigt das Ergebnis für su_{tg} nach Ausführung von Algorithmus 16. Grün dargestellte Kreise repräsentieren exemplarisch Bauteile der Struktureinheit, denen ein Vorgang der Datenbasis TG_{DB} zugeordnet werden konnte (vgl. Abbildung 4.10). Gelb dargestellte Kreise entsprechen den noch zuzuordnenden Voraussetzung eines Vorgangs. Sie referenzieren kein konkretes Bauteil, sondern entsprechen einer Bauteilbeschreibung nach Kapitel 4.2.2.

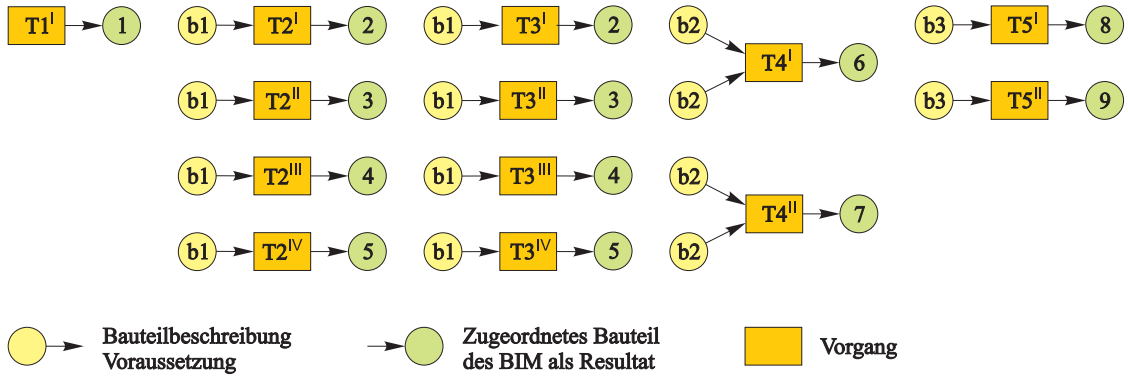


Abbildung 4.11: Menge der Vorgangsgraphen su_{tg} der Struktureinheit nach der Zuordnung von Vorgängen

Schritt 2: Zuordnung von Voraussetzungen zu Vorgängen Die Zuordnung von Voraussetzungen zu den im Schritt 1 identifizierten Vorgängen einer Struktureinheit su_{tg} erfolgt in zwei Stufen. In der ersten Stufe werden die Bauteilobjekte, die Voraussetzungen eines Vorgangs darstellen, mit den Bauteilen der betrachteten Struktureinheit verglichen. Es werden analog zu Schritt 1 Bauteile identifiziert, die den als Voraussetzung modellierten Bauteilen ähnlich sind. Eine Zuordnung der auf diese Weise bestimmten Bauteile ist so nicht in jedem Fall möglich, da diese unter Umständen nicht eindeutig bestimmt werden können. Abbildung 4.12 verdeutlicht das Problem an einem Beispiel. Für das in Abbildung 4.12 dargestellte Beispiel wird angenommen, dass ein Vorgang innerhalb der Datenbasis existiert, der den Einbau der in blau dargestellten Fensterfront repräsentiert und dieser in Schritt 1 zugeordnet wurde. Weiterhin wird angenommen, dass der Vorgang als Voraussetzung eine Wand hat, deren Beschreibung auch derjenigen entspricht, in dem sich die Fensterfront im BIM befindet. Ein entsprechender Vergleich des Bauteilobjekts der Voraussetzung des Vorgangs mit den Bauteilen der teilweise dargestellten Struktureinheit identifiziert alle gelb dargestellten Wände als ähnlich. Demnach sind alle auf diese Weise bestimmten Wände mögliche Kandidaten, für die Zuordnung als Voraussetzung dieses Vorgangs. Die offensichtlich richtige Wand ist jedoch die, in der sich die blau dargestellte Fensterfront befindet. Um diese automatisch zu bestimmen, werden in einer zweiten Stufe alle Kandidaten einer Voraussetzung des betrachteten Vorgangs einer geometrischen Prüfung unterzogen.

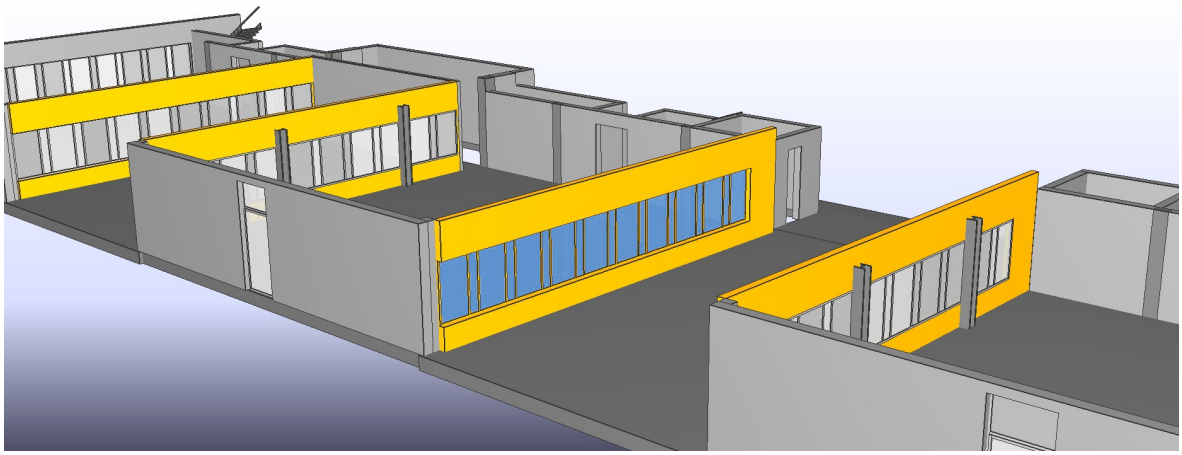


Abbildung 4.12: Geometrische Nähe bei der Bestimmung von Voraussetzungen eines Vorgangs

Algorithmus 17: isIntersect($bounds1, bounds2, \delta$)
Verschneidungstest zweier Bounding-Box Objekte

input : $bounds1(lower.x, lower.y, lower.z, upper.x, upper.y, upper.z)$, $bounds2(lower.x, lower.y, lower.z, upper.x, upper.y, upper.z)$, δ
output : **true** wenn $bounds1$ und $bounds2$ sich schneiden, sonst **false**

1 **if** $(bounds1.upper.x{+}\delta > bounds2.lower.x \wedge bounds2.upper.x > bounds1.lower.x{-}\delta \wedge bounds1.upper.y{+}\delta > bounds2.lower.y \wedge bounds2.upper.y > bounds1.lower.y{-}\delta \wedge bounds1.upper.z{+}\delta > bounds2.lower.z \wedge bounds2.upper.z > bounds1.lower.z{-}\delta)$ **then return** true;
2 **else return** false

Dies geschieht aufgrund der Annahme, dass Bauteile, die zwingend voneinander abhängig sind, eine geometrische Nähe aufweisen und dieser Umstand bei der Definition der Vorgangsgraphen vom Anwender berücksichtigt wird. Eine automatische Bestimmung von Voraussetzungen eines Vorgangs kann anderenfalls nicht erfolgen.
Für die geometrische Prüfung wird zunächst ein Verschneidungstest des Resultats mit den als mögliche Kandidaten bestimmten Bauteilen der betrachteten Struktureinheit durchgeführt. Die rechnerisch einfachste Möglichkeit einer solchen Prüfung ist ein Verschneidungstest der Bounding-Boxen. Dazu wird die Bounding-Box des Resultats um δ Einheiten vergrößert und der Verschneidungstest mit den Bounding-Boxen aller möglichen Voraussetzungen durchgeführt. Der Verschneidungstest kann nach Algorithmus 17 für achsenparallele Bounding-Boxen durchgeführt werden. Es wird vorausgesetzt, dass die Bounding-Boxen $bounds1$ und $bounds2$ im selben Koordinatensystem definiert sind.

Anmerkung: Bei den an dieser Stelle beschriebenen Bounding-Boxen handelt es sich nicht um jene der Bauteilbeschreibungen für die Bestimmung der Ähnlichkeit nach Kapitel 4.2.2. Für die geometrische Prüfung handelt es sich um die Bounding-Boxen von Bauteilen des BIM. Die Koordinaten der Bounding-Boxen befinden sich somit im globalem Koordinatensystem des BIM.

Mögliche Bounding-Boxen von Voraussetzungen, die keine Verschneidung mit der Bounding-Box des Resultats des betrachteten Vorgangs aufweisen, entfallen. Bei ungünstiger Geometrie eines oder beider Körper ist ein Verschneidungstest mittels Bounding-Box jedoch sehr ungenau. Abbildung 4.13 zeigt schematisch ein Beispiel für ein solches Problem. Die Stütze S3 würde in diesem Fall fehlerhaft als Voraussetzung des Vorgangs identifiziert.

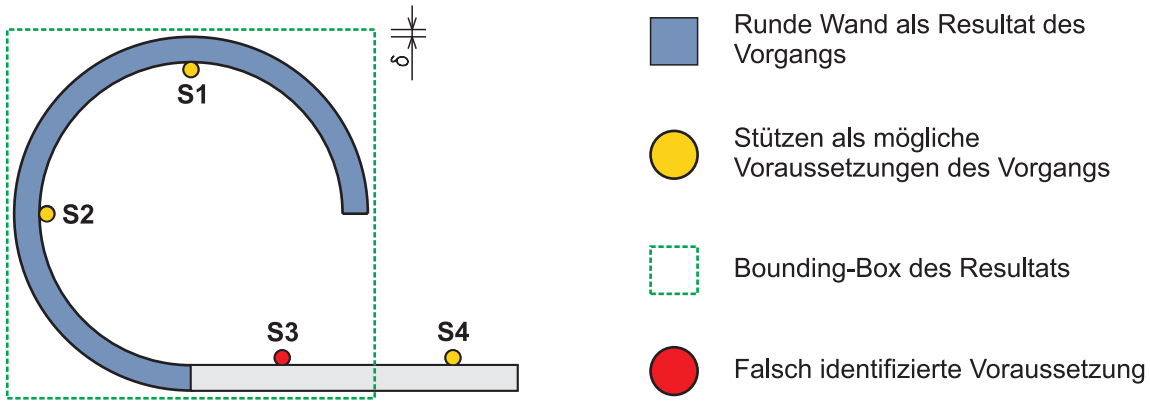


Abbildung 4.13: Verschneidungstest mittels Bounding-Box

Das Ergebnis kann durch den Einsatz genauerer Verfahren, beispielsweise zur Bestimmung des minimalen Abstands zwischen Resultat und den möglichen Voraussetzungen oder einem Verschneidungstest mit der um δ vergrößerten Geometrie des Resultats, verbessert werden. Bei Vorgabe eines entsprechenden Toleranzmaßes für den minimalen Abstand δ werden nur die Stützen S1 und S2 identifiziert (siehe Abbildung 4.14).

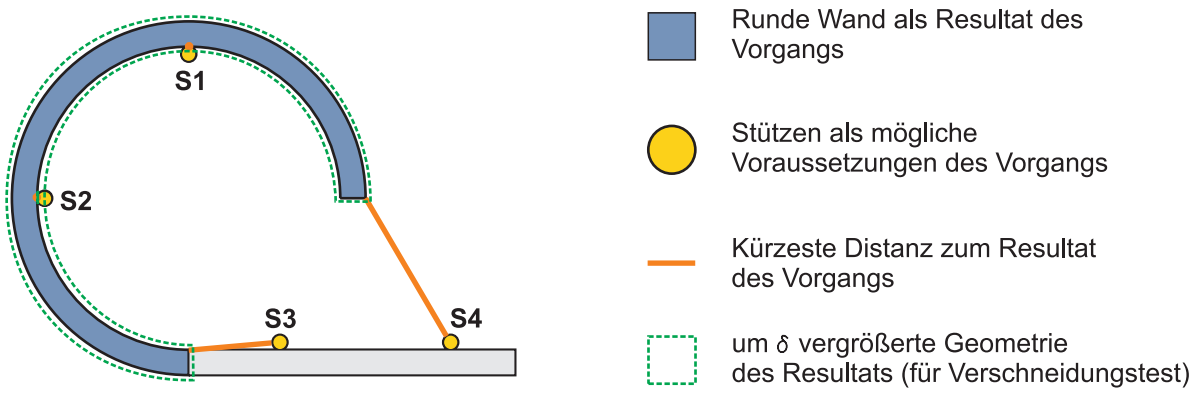


Abbildung 4.14: Minimale Distanz zwischen zwei Körpern

Effiziente Algorithmen für die Bestimmung der Distanz zwischen konvexen Polyedern

und nicht konvexen Polyedern sind in [Dyllong u. a. 1999] bzw. [Kawachi u. Suzuki 2000] dargestellt. Ein genauer Verschneidungstest zweier dreidimensionaler Körper ist in [Möller 1997] dargestellt oder kann durch Anpassung der in [Hubbard 1990] oder [Laidlaw u. a. 1986] dargestellten Algorithmen erreicht werden. Alle genannten Algorithmen setzen Oberflächen voraus, die durch Dreiecke beschrieben werden. Die Anwendung eines Verfahrens wird somit von der verfügbaren Geometrie der Bauteile des BIM bestimmt und muss gegebenenfalls entsprechend überführt werden.
Formal kann der minimale Abstand zweier geometrischer Körper min_{dist} nach Formel 4.24 ausgedrückt werden.

$$min_{dist}(b_{geom_i}, b_{geom_j}) := min(|p - q|)$$

$$\text{mit } b_{geom_i} := \{p \mid p \text{ ist ein Punkt innerhalb oder auf der Oberfläche von } b_i\}$$
$$\text{und } b_{geom_j} := \{q \mid q \text{ ist ein Punkt innerhalb oder auf der Oberfläche von } b_j\} \tag{4.24}$$

Liegt eine mögliche Voraussetzung innerhalb der vorzugebenden Toleranzgrenze δ, wird das auf diese Weise identifizierte Bauteil dem Vorgang als Voraussetzung zugeordnet und steht für eine weitere Zuordnung als Voraussetzung für diesen Vorgang nicht mehr zur Verfügung.
Ist die Anzahl der auf diese Weise zuzuordnenden Voraussetzungen größer oder kleiner als die Anzahl der Voraussetzungen des Vorgangs, muss dieser Konflikt durch den Anwender anhand einer manuelle Zuordnung gelöst werden.
Algorithmus 18 zeigt den zugehörigen Algorithmus als Pseudocode.

Beschreibung Algorithmus 18: Als Eingangswerte dienen die zu betrachtende Struktureinheit *su*, der Ähnlichkeitsgrenzwert lim_{sim} und die Toleranzgrenze δ. Es wird vorausgesetzt, dass Algorithmus 16 für die Bestimmung der Vorgänge bereits ausgeführt wurde und die Bedingung $su_{tg} \neq \varnothing$ erfüllt ist.
Für jeden Vorgangsgraphen $currentTG \in su_{tg}$ wird für jede Beschreibung eines Bauelements $currentBE_{su}$, das eine Voraussetzung des Vorgangs *currentTG* ist, nach ähnlichen Bauteilen in der Menge der Bauteile su_b der betrachteten Struktureinheit gesucht. Falls ein solches Bauteil identifiziert wird, das größer oder gleich dem gegebenen Ähnlichkeitsgrenzwert lim_{sim} ist, wird es der Menge *possiblePre* als mögliche Voraussetzung für die gerade betrachtete Bauteilbeschreibung *currentPre* zugeordnet (vgl. Zeilen 7-11 in Algorithmus 18).
Im Anschluss wird geprüft, ob ein Bauteil, das als mögliche Voraussetzung für *currentPre* bestimmt wurde, eine geometrische Nähe innerhalb der Toleranzgrenze δ aufweist und dem aktuellen Vorgangsgraphen *currentTG* noch nicht zugeordnet wurde. Ist dies der Fall, wird geprüft, ob dem aktuellen Vorgangsgraphen *currentTG* für *currentPre* bereits ein Bauteil zugeordnet wurde. Ist dies der Fall, ist das Bauteil *currentPossiblePre* überzählig. Es würde demnach die Kriterien einer Zuordnung erfüllen, kann aber nicht

Algorithmus 18: determinePrerequisites(su, lim_{sim}, δ)
Zuordnung von Voraussetzungen zu Vorgängen

input : $su(su_n, su_r, su_{guid}, su_b, su_{tg}, su_s)$, lim_{sim}, δ
output : Menge der Vorgangsgraphen $su_{tg} \in su$

```
1  if su_tg ≠ ∅ then return;
2  foreach currentTG(T, B, P, R) ∈ su_tg do
3      resultBE := currentRes | r(currentTask, currentRes) ∈ R;
4      leftOverBE := ∅;
5      foreach p(currentPre, currentTask) ∈ P do
6          possiblePre := ∅;
7          foreach currentBE_su ∈ su_b do
8              if sim_o(currentBE_su, possiblePre) ≥ lim_sim then
9                  possiblePre := possiblePre ∪ {currentBE_su}
10             end
11         end
12         foreach currentPossiblePre ∈ possiblePre do
13             check = false;
14             if min_dis(currentPossiblePre, resultBE) ≤ δ)
               ∧currentPossiblePre ∉ B then
15                 if check = true then
                   leftOverBE := leftOverBE ∪ {currentPossibleBE};
16                 else
17                     check = true;
18                     leftOverBE := leftOverBE \ {currentPossiblePre};
19                     P \ {(currentPre, currentTask)};
20                     B \ currentPre;
21                     B := B ∪ {currentPossiblePre};
22                     P := P ∪ {(currentPossiblePre, currentTask)};
23                 end
24             end
25         end
26     end
27     if |leftOverBE| > 0 then manuelle Zuordnung der Voraussetzungen für
       currentTG notwendig
28 end
```

mehr zugeordnet werden. Es besteht jedoch die Möglichkeit, dass dieses Bauteil einer weiteren Vorbedingung des Vorgangsgraphen *currentTG* zugeordnet werden kann. Es wird zunächst in die Menge *leftOverBE* eingetragen. Wurde der Voraussetzung *currentPre* noch kein Bauteil zugeordnet, wird das Bauteil *currentPossiblePre* aus der Menge *leftOverBE* entfernt und als Voraussetzung für *currentPre* in den Vorgangsgraphen *currentTG* eingefügt (vgl. Zeilen 19-22 in Algorithmus 18).
Ist der Betrag der Menge *leftOverBE* größer 0, konnten nicht alle Vorbedingungen eindeutig zugeordnet werden. In diesem Fall ist eine manuelle Zuordnung bzw. Überprüfung der Voraussetzungen durch den Anwender notwendig.
Ist eine manuelle Zuordnung nicht möglich, muss ein neuer Vorgang definiert und in die Datenbasis TG_{DB} eingefügt werden. Dies ist der Fall, wenn nach der manuellen Zuordnung der Betrag der Differenz der Menge der Bauteile aller Vorgangsgraphen su_{tg}, die Voraussetzung eines Vorgangs sind, und der Menge der Bauteile der Struktureinheit su_b größer 0 ist.

$$su_{p_{left}} := \{b \in B_{TG} \mid p(b,t) \in P_{TG}\} \setminus su_b$$

$su_{p_{left}}$ ist die Menge der Voraussetzungen aller Vorgangsgraphen su_{tg}, für die kein Bauteil identifiziert werden konnte

$$\text{mit } B_{TG} := \left\{ \bigcup_{i=1}^{|su_{tg}|} B_i \mid TG_i(T_i, B_i, P_i, R_i) \in su_{tg} \right\}$$
$$\text{und } P_{TG} := \left\{ \bigcup_{j=1}^{|su_{tg}|} R_i \mid TG_i(T_i, B_i, P_i, R_i) \in su_{tg} \right\} \tag{4.25}$$

Wurde ein neuer Vorgangsgraph tg_{new} definiert und in die Datenbasis TG_{DB} eingefügt, sind für die Algorithmen 16 und 18 für die betrachtete Struktureinheit *su* erneut auszuführen.

Beispiel: Abbildung 4.15 zeigt das Ergebnis für $su_{tg} \in su$ nach Ausführung von Algorithmus 16. Den Voraussetzungen der nach Algorithmus 16 bestimmten Vorgänge wurden nun ebenfalls konkrete Bauteile der Struktureinheit zugeordnet (vgl. Abbildung 4.10).

Verknüpfungsphase

Während der Verknüpfungsphase werden alle innerhalb der *Zuordnungsphase* ermittelten Vorgangsgraphen einer Struktureinheit su_{tg} zusammengeführt. Formal gesehen handelt es sich um die Vereinigung aller bisher bestimmten Vorgangsgraphen, wobei deren Voraussetzungs- und Ergebnisknoten nun durch konkrete Bauteile der Struktureinheit su_b repräsentiert werden (siehe Formel 4.26).

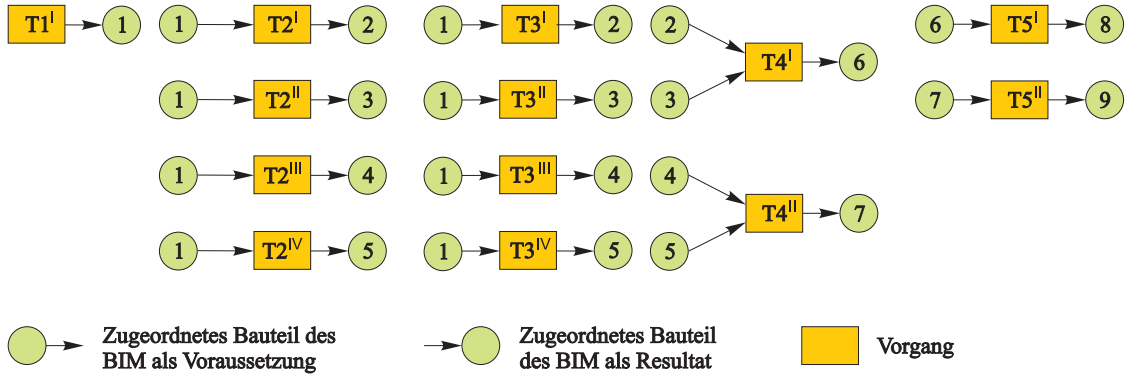


Abbildung 4.15: Menge der Vorgangsgraphen su_{tg} der Struktureinheit nach der Zuordnung von Vorgängen und Voraussetzungen

$$su_s := \bigcup_{i=1}^{|su_{tg}|} su_{tg_i} \mid su_{tg_i}(T_i, B_i, P_i, R_i) \in su_{tg} \tag{4.26}$$

Als Ergebnis ergibt sich ein erster Bauablauf für die Vorgänge der betrachteten Struktureinheit su. Dieser entsteht aufgrund der durch die Vereinigung der Strukturgraphen su_{tg} resultierenden Verknüpfung identischer Bauteile der Vorbedingungen und Resultate nach These 5 (siehe Seite 28). Algorithmus 19 zeigt die Vorgehensweise.

Algorithmus 19: *mergeTaskGraphs*(su)
Kombination der Vorgangsgraphen

input : $su(su_n, su_r, su_{guid}, su_b, su_{tg}, su_s)$
output : bipartiter Graph $su_s \in su$

```
1 su_s := (T, B, P, R);
2 T := ∅; B := ∅; P := ∅; R := ∅;
3 foreach currentTG(T_current, B_current, P_current, R_current, ) ∈ su_tg do
4 |   T := T ∪ T_current; B := B ∪ B_current;
5 |   P := P ∪ P_current; R := R ∪ R_current;
6 end
```

Beispiel: Abbildung 4.16 zeigt als Ergebnis von Algorithmus 19 den bipartiten Graphen $su_s \in su$. Identische Bauteile existieren aufgrund der Vereinigung aller Vorgangsgraphen su_{tg} nur noch ein mal. Es ergibt sich die Reihenfolge der Vorgänge $T1^I$ bis $T5^I$.

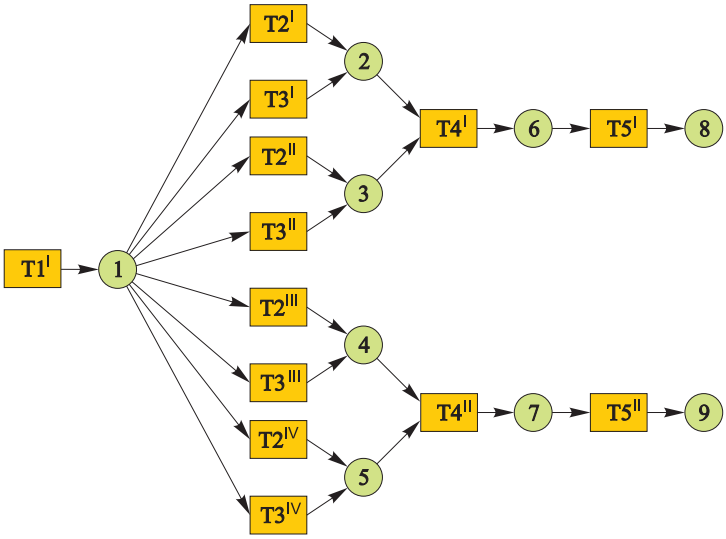


Abbildung 4.16: Bipartiter Graph su_s nach der Vereinigung aller Vorgangsgraphen $su_{tg} \in su$

Konfliktlösungsphase

Nach dem erfolgreichen Durchlaufen der Zuordnungs- und Verknüpfungsphase liegt als Ergebnis ein bipartiter Graph su_s vor. Der auf die vorgeschlagene Weise generierte Bauablauf ist jedoch im Sinne eines Workflowgraphen nicht zwangsläufig korrekt. Ziel der Konfliktlösungsphase ist die Überführung des bipartiten Graphen su_s in einen strukturell korrekten Workflowgraphen für die Struktureinheit su.

Zusammenfassung der Struktureinheit Für einen Workflowgraphen wird gefordert, dass er einen Start- und einen Endknoten besitzt. In einem ersten Schritt werden alle bestehenden Startknoten des Graphen su_s an einen neuen Startknoten $S \in T$ und alle zu diesem Zeitpunkt bestehenden Endknoten an einen neuen Endknoten $E \in T$ angebunden. Dies bedeutet, dass bisherigen Startknoten $s \in T$ über einen neuen Knoten $b_i \in B$ und alle bisherigen Startknoten $s \in B$ direkt durch Kanten angebunden werden. Für den neuen Endknoten E gilt dies analog. Algorithmus 20 beschreibt die zugehörige Vorgehensweise.

Beispiel: Abbildung 4.17 zeigt als Ergebnis von Algorithmus 20 den bipartiten Graphen $su_s \in su$. Der bisherige Startknoten $T1^I$ wurde über die Kanten $p(ps, T1^I)$, $r(S, ps)$ und den Scheinknoten ps an den neuen Startknoten S angebunden. Der neue Endknoten E wurde über die Kanten $p(8, E)$ und $p(9, E)$ an die bisherigen Endknoten 8 und 9 angeschlossen.

Algorithmus 20: *createStartEndNodes*(su)
Zusammenfassung der Struktureinheit

input : $su(su_n, su_r, su_{guid}, su_b, su_{tg}, su_s)$
output : bipartiter Graph $su_s \in su$

```
1  currentStartNodes := getStartNodes (su_s);
2  currentEndNodes := getEndNodes (su_s);
3  T := T ∪ {new startNode | su_s(T, B, P, R)};
4  T := T ∪ {new endNode | su_s(T, B, P, R)};
5  foreach currentStartNode ∈ currentStartNodes do
6  |  if currentStartNode ∈ T then
7  |  |  B := B ∪ {new pseudoNode};
8  |  |  R := R ∪ {(startNode, pseudoNode)};
9  |  |  P := P ∪ {(pseudoNode, currentStartNode)}
10 |  end
11 |  else
12 |  |  R := R ∪ {(startNode, currentStartNode)};
13 |  end
14 end
15 foreach currentEndNode ∈ currentEndNodes do
16 |  if currentEndNode ∈ T then
17 |  |  B := B ∪ {new pseudoNode};
18 |  |  R := R ∪ {(currentEndNode, pseudoNode)};
19 |  |  P := P ∪ {(pseudoNode, endNode)}
20 |  end
21 |  else
22 |  |  P := P ∪ {(currentEndNode, endNode)};
23 |  end
24 end
```

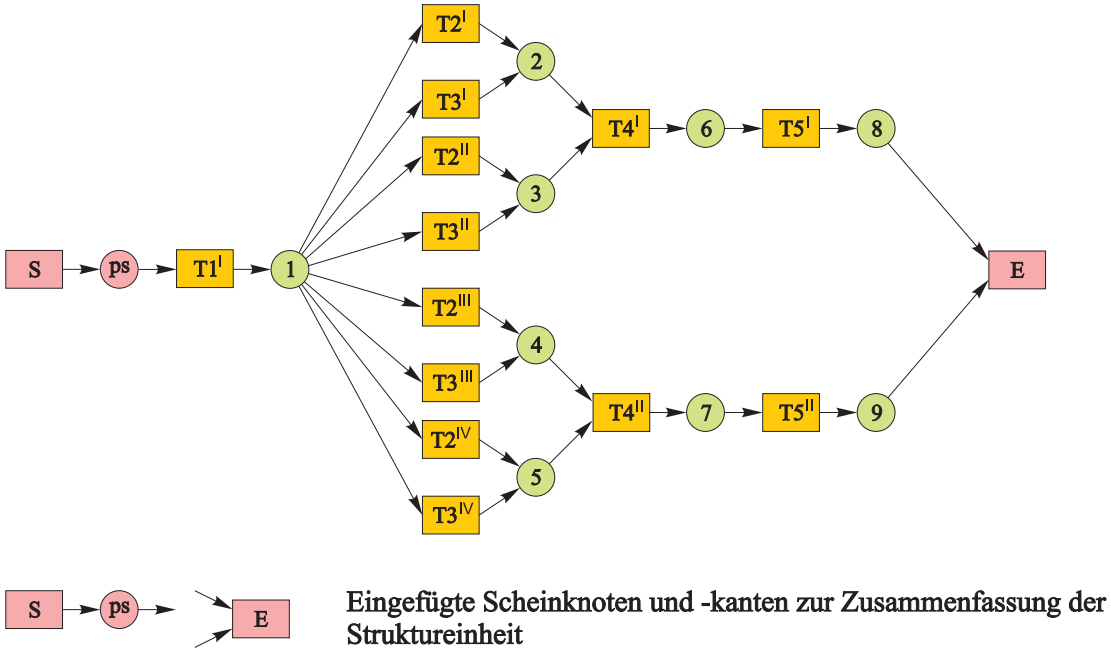


Abbildung 4.17: Ergebnis der Zusammenfassung des Graphen su_s der Struktureinheit

Konfliktlösung Um eventuelle Konflikte des Graphen su_s aufzulösen, müssen diese zunächst erkannt werden. Dazu wird in einem nächsten Schritt der bipartite Graph su_s als Workflowgraph betrachtet und mittels *well handled test* nach Kapitel 3.4.4 seine strukturelle Integrität geprüft. Das Ergebnis des Algorithmus *getConflictPares*(su_s) liefert alle Konfliktpaare des bipartiten Graphen su_s.

Beispiel: Für das Beispiel ergeben sich die Konfliktpaare $(1, T4^I)$, $(1, T4^{II})$ und $(1, E)$, insofern der XOR-Split-Knoten 1 mit den AND-Join-Knoten $T4^I$, $T4^{II}$ bzw.E synchronisiert wird (siehe Abbildung 4.18).

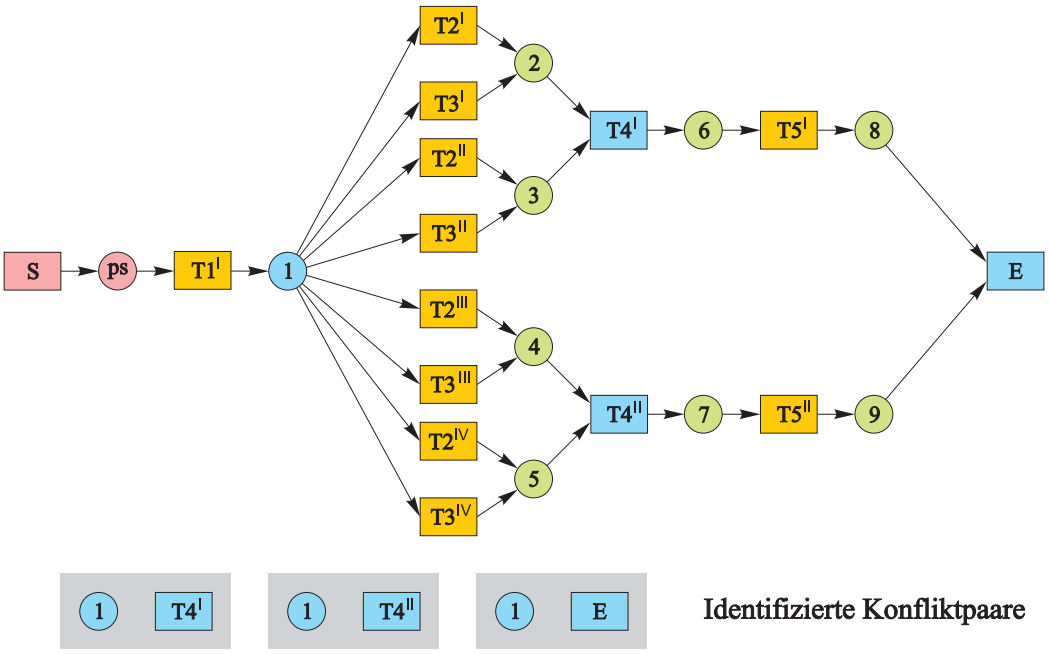


Abbildung 4.18: Identifizierte Konfliktpaare des Graphen su_s

Aufgrund der Definition, dass ein Vorgang jeweils nur ein Resultat liefern kann (siehe Kapitel 4.2.4), ist die Synchronisation eines XOR-Split-Knotens durch einen AND-Join die einzige Klasse von strukturellen Fehlern, die bei Einhaltung aller genannten Bedingungen an dieser Stelle auftreten kann. Eine falsche Synchronisation eines XOR-Splits ist gleichbedeutend mit dem Vorhandensein paralleler Vorgänge bzw. Vorgangsketten.

Beschreibung Algorithmus 21: Die Eingangswerte des Algorithmus *solveConflicts* sind der bipartite Graph su_s nach der Ausführung von Algorithmus 19, der Index des vorherigen Durchlaufs *currentStep* sowie die maximale Anzahl von Durchläufen *maxSteps*. Die Übergabe der beiden Indexvariablen *currentStep* und *maxSteps* dient der sicheren Terminierung des Algorithmus bei unerwartet auftretenden Fehlern, da er nach der Korrektur eines Konfliktpaars rekursiv aufgerufen wird, um verbliebene Konflikte zu lösen. Die Anzahl der zu korrigierenden Fehler und somit der maximal zu korrigierenden Konflikte kann dabei höchstens der Anzahl der Konfliktpaare beim ersten Aufruf des Algorithmus entsprechen. Die Indexvariable *maxSteps* wird nach einem externen Aufruf mit dieser Anzahl initialisiert. *currentStep* wird vor jedem Durchlauf inkrementiert. Für einen externen Aufruf des Algorithmus ist für *currentStep* und *maxSteps* der Wert -1 zu übergeben.
Existiert wenigstens ein Konfliktpaar $c_1(a, b)$, so wird geprüft, ob Knoten a ein XOR-Split ist. Ist dies der Fall, werden alle Nachfolger $successors := \{a_{suc_k} \mid a_{suc_k} \in t_N(a)\}$ des Knotens a ermittelt. Dies dient der Ermittlung aller von a abgehenden Kanten (a, a_{suc_k}), die für die Konfliktlösung behandelt werden müssen (vgl. Zeile 8 in Algorithmus 21).
Danach werden alle zu Knoten a korrespondierenden XOR-Join-Knoten *joins* ermittelt (siehe Kapitel 3.4.1). Existieren solche Knoten, wird ein Scheinknoten t in die Menge der Vorgänge $T \in su_s$ eingefügt und über die Kante (a, t) angebunden. Anschließend wird die Menge der knotendisjunkten Wege zwischen a und jedem Knoten *currentJoin* ermittelt. Existieren solche Wege, müssen diese über einen neuen XOR-Split a' an den Vorgang t angebunden werden. Die bisherige Kante $(node_0, node_1)$ entfällt und wird aus der Menge der Vorbedingungen P ausgetragen. Die ehemalige Kante wurde bearbeitet und somit wird der Knoten $node_1$ aus der Menge der zu bearbeitenden Nachfolger *successors* ausgetragen (vgl. Zeilen 13 - 19 in Algorithmus 21)
Enthält die Menge der zu bearbeitenden Nachfolger *successors* noch Knoten, handelt es sich um parallele Vorgänge bzw. Vorgangsketten. Jeder dieser Knoten *currentSuccessor* muss demnach mittels der Kanten $(t, a') \in R$ und $(a', currentSuccessor) \in P$ über einen neuen Knoten $a' \in B$ angebunden werden. Die bisherige Kante $(a, currentSuccessor) \in P$ entfällt (vgl. Zeilen 24 - 32 in Algorithmus 21).
Ist die maximale Anzahl an möglichen Durchläufen noch nicht erreicht, erfolgt der rekursive Aufruf des Algorithmus mit den Parametern su_s, *currentStep* und *maxSteps*.

> **Anmerkung:** Ist nach Terminierung von Algorithmus 21 die Anzahl der Konfliktpaare ungleich 0, ist die Struktur des Workflowgraphen inkonsistent. In diesem Fall ist eine manuelle Korrektur des Workflowgraphen durch den Anwender notwendig.

Algorithmus 21: *solveConflicts*(su_s, *currentStep*, *maxSteps*)
Zusammenfassung der Struktureinheit

input : $su_s(T, B, P, R)$, *currentStep*, *maxSteps*
output : Workflow Graph $su_s \in su$

1 $conflictsPairs(c_i, c_{i+1}, \ldots, c_{|conflictPairs|}) :=$ `getConflictPairs`(su_s);
2 **if** $maxSteps = -1$ **then** $maxSteps := |conflictPairs|$;
3 $currentStep := currentStep + 1$; $xorSplits :=$ `getXorSplits`(su_s);
4 $t =$ **new** *Task*; $successors := \varnothing$;
5 **if** $|conflictPairs| \geq 1$ **then**
6 $currentConflict(a, b) := c_1$;
7 **if** $currentConflict \in xorSplits$ **then**
8 $successors := t_N(a)$;
9 $joins :=$ `getJoinsFromSplit`(a);
10 **if** $|joins| \geq 1$ **then** $T := \cup\{t\}$; $P := P \cup \{(a, t)\}$;
11 **foreach** $currentJoin \in joins$ **do**
12 $paths :=$ `getNodeDisjointPaths`(su_s, a, $currentJoin$);
13 **if** $|paths| > 1$ **then**
14 $a' :=$ **new** *node*; $B := B \cup \{a'\}$;
15 **foreach** $currentPath(node_0, \ldots, node_{|currentPath-1|}) \in paths$ **do**
16 $P := P \setminus \{(node_0, node_1)\}$;
17 $P := P \cup \{(a', node_1)\}$;
18 $R := R \cup \{(t, a')\}$;
19 $successors \setminus \{node_1\}$;
20 **end**
21 **end**
22 **end**
23 **end**
24 **foreach** $currentSuccessor \in successors$ **do**
25 $a' :=$ **new** *node*;
26 **if** $t \notin T$ **then**
27 $T := T \cup \{t\}$; $P := P \cup \{(a', t)\}$
28 **end**
29 $P := P \setminus \{(a, currentSuccessor)\}$;
30 $B := B \cup \{a'\}$;
31 $P := P \cup \{(a', currentSuccessor)\}$;
32 $R := R \cup \{(t, a')\}$;
33 **end**
34 **end**
35 **else return**;
36 **if** $currentStep \leq maxSteps$ **then**
37 `solveConflicts` (su_s, *currentStep*, *maxSteps*);
38 **end**

Beispiel: Abbildung 4.19 zeigt das endgültige Ergebnis nach Durchlaufen der Konfliktlösungsphase für das benannte Beispiel nach Algorithmus 21.

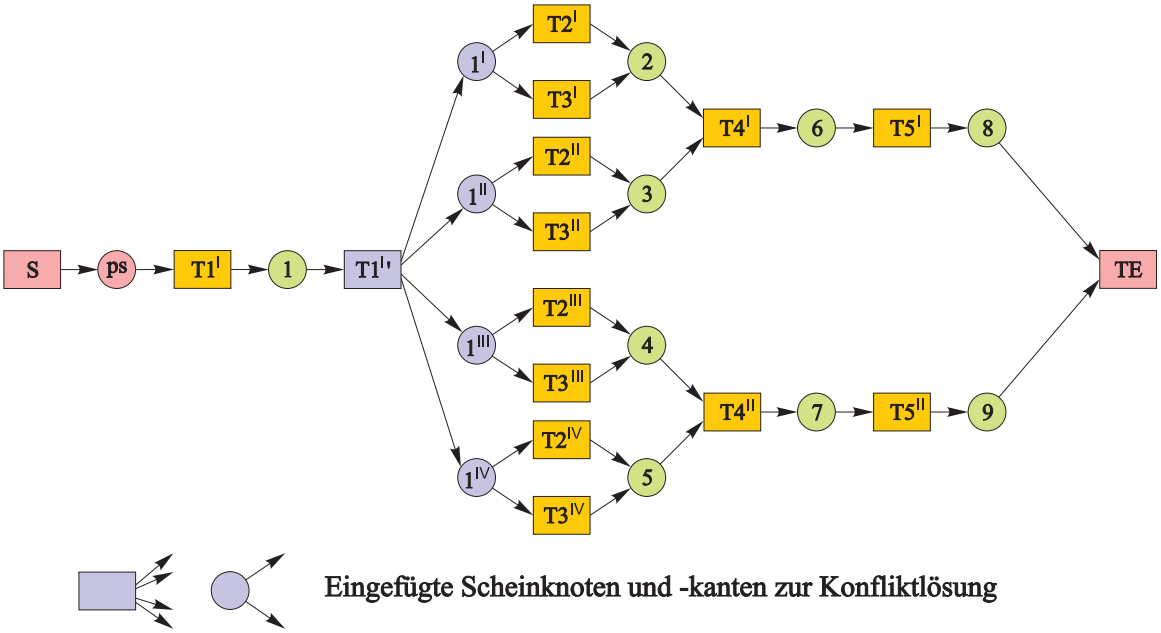


Abbildung 4.19: Ergebnis für den Graphen su_s nach Durchlaufen der Konfliktlösungsphase

4.3.3 Kopplungsphase

Innerhalb der Kopplungsphase wird der Gesamtbauablauf S in Form eines Workflowgraphen generiert. Für die Veranschaulichung der in den Algorithmen 22 und 23 dargestellten Vorgehensweise dient das in Abbildung 4.20 dargestellte Beispiel. Es zeigt fünf Workflowgraphen su_s der Struktureinhieten su_1 bis su_5 sowie den Strukturgraphen SG, der die Reihenfolge der Struktureinheiten definiert.

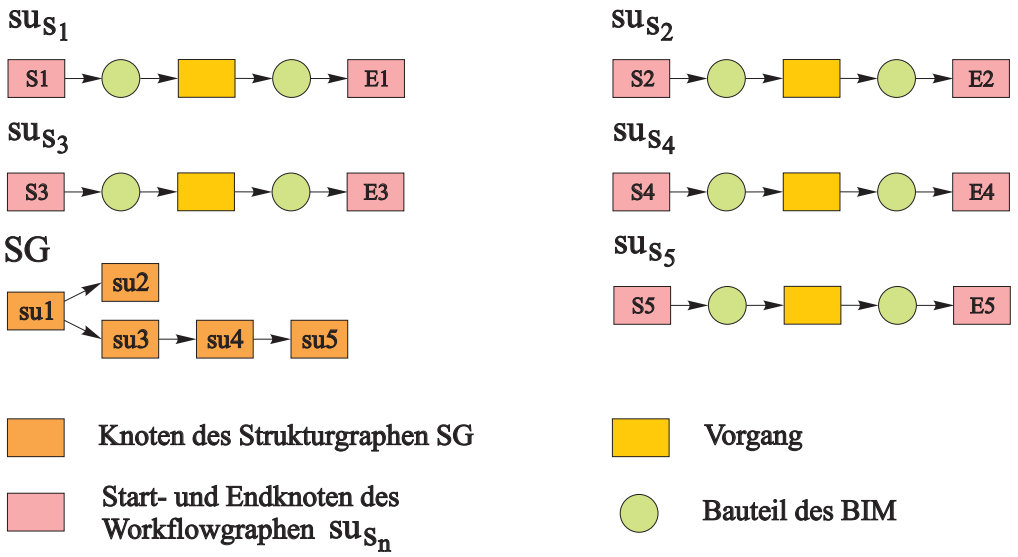


Abbildung 4.20: Beispiel für 5 Struktureinheiten su_{s_1} bis su_{s_5} und Strukturgraph SG

Algorithmus 22: *linkStructureUnits*(SU, SG)
Kopplung der Workflowgraphen aller Struktureinheiten

input : $SU, SG(SU, O)$
output : S

```
1  new WorkflowGraph S(T_s, B_s, P_s, R_s);
2  new Task t_start;
3  T_s := T_s ∪ {t_start};
4  new Task t_end;
5  T_s := T_s ∪ {t_end};
6  SU_start := {su_i ∈ SU | g_V(su_i) = 0};
7  foreach su_start ∈ SU_start do
8  |  appendUnit (t_start, t_end, su_start, SG, S);        // Algorithmus 23
9  end
10 return S;
```

Beschreibung Algorithmus 22: Als Eingangsparameter von Algorithmus 22 dienen die Menge der Struktureinheiten SU sowie der Strukturgraph SG. Zunächst wird ein neuer Workflowgraph S erzeugt und dessen Startknoten t_{start} und Endknoten t_{end} definiert und in die Menge der Vorgänge T eingefügt. Anschließend wird die Menge aller Startknoten im Strukturgraph SG bestimmt. Die Workflowgraphen su_s aller auf diese Weise bestimmten Struktureinheiten su_{start} werden unmittelbar an den Startknoten t_{start} angebunden. Dies geschieht durch den Aufruf von **appendUnit**(t_{start}, t_{end}, su_{start}, SG, S) - siehe Algorithmus 23.

Beschreibung Algorithmus 23: Als Eingangswerte des Algorithmus 23 dienen der Vorgängerknoten $predecessor \in T_s$ im Strukturgraphen SG, an den der Workflowgraph $su_s \in su$ gekoppelt werden soll, der Parameter t_{end}, der dem Endknoten des Workflowgraphen S entspricht, der zugehörige Strukturgraph SG und der Workflowgraph S selbst.
In einem ersten Schritt wird der zu koppelnde Workflowgraph su_s der Struktureinheit su geklont und als su'_s in den Workflowgraphen S eingefügt. Ein Klonen ist notwendig, da die innerhalb der Kopplungsphase zusammenzuführenden Workflowgraphen identische Bauteile als Knoten beinhalten können. Würde dieser Schritt unterlassen, werden diese Knoten als identische Knoten betrachtet und die Workflowgraphen davon betroffener Struktureinheiten würden sich über ihre Kanten verbinden. Die Bindung zu den Bauteilen des BIM ist durch das geforderte Attribut $a_n :=$"id" auch nach dem Klonen existent (vgl. Kapitel 4.2.2). Anschließend wird der übergebene Vorgängerknoten *predecessor* über den Scheinknoten *pseudoNode* an den Startknoten des Workflowgraphen su'_s angebunden und der Endknoten $su_{b_{end}}$ des Workflowgraphen su_s bestimmt. Der Aufruf **getStartNodes**(su'_s) bzw. **getEndNodes**(su'_s) liefert dabei innerhalb der jeweiligen Ergebnismenge einzig den gesuchten Start- bzw. Endknoten, da alle Workflowgrahen $su_s \in su$ nach Durchlaufen der Konfliktlösungsphase nur einen Start- und Endknoten besitzen.
Im nächsten Schritt werden alle Struktureinheiten SU_{suc} ermittelt, die su im Struktur-

Algorithmus 23: *appendUnit*(*predecessor*, *endNode*, *su*, *SG*, *S*)
Kopplung einer Struktureinheit an seinen Vorgänger

input : $predecessor$, t_{end}, $su(su_n, su_r, su_{guid}, su_{tg}, su_s)$, $SG(SU, O)$, $S(T_s, B_s, P_s, R_s)$

1 // Eintragen des Workflowgraphen $su_s \in su$ in S
2 $su'_s :=$ deepClone (su_s);
3 $S := S \cup su'_s$;
4 // Anbinden von su'_s an den Endknoten seines Vorgängers *predecessor*
5 $B_s := B_s \cup \{$**new** BuildingElement *pseudoNode*$\}$;
6 $R_s := R_s \cup \{(predecessor, pseudoNode)\}$;
7 $P_s := P_s \cup \{(pseudoNode,$ **single Node in** $\{$getStartNodes$(su'_s)\})\}$;
8 // Ermitteln des Endknotens von su'_s
9 $su_{b_{end}} :=$ **single Node in** $\{$getEndNodes$(su'_s)\})\}$;
10 // Ermitteln aller nachfolgenden Struktureinheiten von $su \in SG$
11 $SU_{suc} := \{su_i \in SU \mid g_N(su) = 0\}$ für $SG(SU, O)$;
12 // Existiert kein Nachfolger, su an den Endknoten t_{end} anschließen
13 **if** $|SU_{suc}| = 0$ **then**
14 $\quad B_s := B_s \cup \{$**new** BuildingElement *pseudoNode*$\}$;
15 $\quad R_s := R_s \cup \{(pseudoNode, t_{end})\}$;
16 $\quad P_s := P_s \cup \{(su_{b_{end}}, pseudoNode)\}$;
17 **end**
18 // Sonst, jeden Nachfolger von $su \in SG$ an den Endknoten $su_{b_{end}}$ anschließen
19 **else**
20 $\quad$ **foreach** *successor* $\in SU_{suc}$ **do**
21 $\quad\quad$ appendUnit $(su_{b_{end}}, t_{end}, successor, SG, S)$;
22 $\quad$ **end**
23 **end**

graphen SG folgen. Hat su keinen Nachfolger, wird der Endknoten $su_{b_{end}}$ über einen Scheinknoten an den Endknoten t_{end} des Workflowgraphen S gekoppelt. Sind nachfolgende Struktureinheiten vorhanden, werden diese über den rekursiven Aufruf des Algorithmus gekoppelt.

Beispiel: Abbildung 4.21 zeigt das Resultat für den Workflowgraphen S nach Durchlaufen der Kopplungsphase.

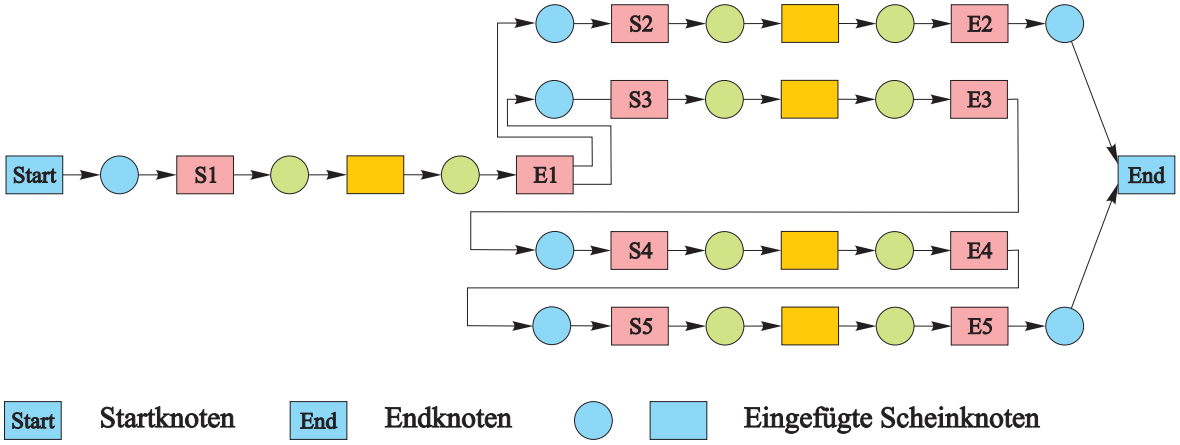


Abbildung 4.21: Resultat der Kopplungsphase für das Beispiel nach Abbildung 4.20

4.3.4 Gesamtablauf der Generierung

Algorithmus 24 fasst die bisher vorgestellten Algorithmen für die Generierung als Gesamtablauf für alle Struktureinheiten $su \in SU$ zusammen.

Beschreibung Algorithmus 24: Auf eine ausführliche Beschreibung des Algorithmus wird an dieser Stelle verzichtet. Die Vorgehensweise kann dem in Abbildung 4.8 dargestellten Flussdiagramm entnommen werden.

Algorithmus 24: *generateWorkflow*$(SU, SG, TG_{DB}, lim_{sim}, \delta)$
Generierung des Workflowgraphen für den Gesamtablauf

input : $SU, SG(SU, O), TG_{DB}, lim_{sim}, \delta$
output : S

```
1  foreach su(su_n, su_r, su_guid, su_b, su_tg, su_s) ∈ SU do
2      repeat
3          repeat
4              repeat
5                  resultsLeft := false;
6                  determineResults (su, lim_sim);                    // Algorithmus 16
7                  if |su_rleft| > 0;                                 // Siehe Formel 4.23
8                  then
9                      resultsLeft := true;
10                     ⇒Manuelle Definition oder Adaption eines
                       Vorgangsgraphen;
11                     T_DB := T_DB ∪ new tg;
12                 end
13             until resultsLeft = true;
14             determinePrerequisites (su, lim_sim, δ);               // Algorithmus 18
15             prerequisitesLeft := false;
16             if |su_pleft| > 0;                                     // Siehe Formel 4.25
17             then
18                 ⇒Manuelle Zuordnung von Voraussetzungen;
19                 if |su_pleft| > 0;                                 // Siehe Formel 4.25
20                 then
21                     prerequisitesLeft := true;
22                     ⇒Manuelle Definition eines Vorgangsgraphen;
23                     T_DB := T_DB ∪ new tg;
24                 end
25             end
26         until prerequisitesLeft = true;
27         mergeTaskGraphs (su);                                      // Algorithmus 19
28         createStartEndNodes (su);                                  // Algorithmus 20
29         solveConflicts (su_s, −1, −1);                             // Algorithmus 21
30         conflictsSolved := true;
31         if |getConflictPairs| > 0;                                 // Algorithmus 14
32         then
33             ⇒Manuelle Korrektur des Workflowgraphen
34             if |getConflictPairs| > 0;                             // Algorithmus 14
35             then
36                 ⇒Manuelle Definition eines Vorgangsgraphen
37                 conflictsSolved := false;
38             end
39         end
40     until conflictsSolved = false;
41     return linkStructureUnits (SU, SG);                            // Algorithmus 22
42 end
```

4.4 Diskussion

Innerhalb dieses Kapitels werden die sich auf Basis des entwickelten Modells ergebenden Möglichkeiten hinsichtlich ihrer Anwendung auf den Planungsprozess für Bauabläufe diskutiert.

4.4.1 Parallele Vorgänge und Varianten

Der generierte Bauablauf S kann sowohl parallele Vorgänge und Vorgangsketten als auch Varianten enthalten.

Parallele Vorgänge Parallele Vorgänge oder Ketten von Vorgängen bezeichnen Vorgänge, die im Bauablauf nebeneinander ausgeführt werden können. Sie beginnen an einem AND-Split-Knoten, werden durch einen AND-Join-Knoten wieder synchronisiert und können somit durch Anwendung der entsprechenden Algorithmen 10 und 12 identifiziert werden (siehe Kapitel 3.4.2).
Ob solche Vorgänge oder Vorgangsketten in der Realität parallel, in Gruppen oder sequentiell ausgeführt werden sollen, unterliegt in der Regel der Ressourcenplanung. Eine derartige Optimierung des Workflowgraphen S der Generierung wird vom vorgestellten Modell nicht bzw. nur bedingt unterstützt. Eine Gruppierung und deren sequentielle Ausführung kann jedoch durch die Definition und Verknüpfung entsprechender Struktureinheiten erreicht werden.

Alternative Vorgänge Es handelt sich hierbei um mögliche Alternativen, die in der Realität zur Herstellung ein und desselben Bauteils führen. Als gegenständliches Beispiel kann das Errichten einer Stahlbetonwand betrachtet werden. Die Wand kann beispielsweise als Ortbetonvariante oder auch durch ein vorgefertigtes Bauteil ausgeführt werden.
Varianten können auf Basis der Struktur des Workflowgraphen S erkannt werden. Liefert mehr als ein Vorgang ein identisches Bauteil als Ergebnis, handelt es sich um eine solche Alternative für die Erstellung eines Bauteils. Innerhalb des Workflowgraphen S handelt es sich hierbei um einen XOR-Join-Knoten. Entscheidungsknoten innerhalb des generierten Workflowgraphen S kennzeichnen demnach alternative Ablaufmöglichkeiten. Eine Variante beginnt mit der Entscheidung an einem XOR-Split-Knoten und endet an einem XOR-Join-Knoten, der das Bauteil repräsentiert, für dessen Erstellung mehr als eine Möglichkeit im Bauablauf existiert. Varianten können demnach durch Anwendung der Algorithmen 8 und 12 identifiziert werden (siehe Kapitel 3.4.1 und 3.4.3).

Varianten werden infolge des Modellentwurfs automatisch unterstützt. Enthält die Datenbasis mehr als eine Möglichkeit für die Herstellung einer durch deren Eigenschaften beschriebenen Klasse von Bauteilen, so werden diese Vorgänge automatisch mit den entsprechenden Bauteilen verknüpft und sind dadurch im Bauablauf S enthalten.

4.4.2 Auswahl von Varianten

Die Auswahl von Varianten im Bauablauf kann automatisch unterstützt werden. Durch das Anwenden von Algorithmus 15 können alle Instanzgraphen ausgehend von einem Knoten bestimmt werden. Die Menge der identifizierten Instanzgraphen entspricht dabei allen möglichen Bauabläufen in S ausgehend vom gegebenen Startknoten *startNode*. Durch Anwenden bekannter Methoden der Netzplantechnik wie beispielsweise CPM oder PERT (siehe Kapitel 2.1.2) kann auf Basis der jeweilige Analyse eine entsprechende Auswahl für einen bestimmten Bauablauf erfolgen. Denkbare Szenarien betreffen beispielsweise die Ermittlung des schnellsten oder kostengünstigsten Bauablaufs im Workflowgraphen S.
Grundlage hierfür ist eine entsprechende Bewertung aller Vorgänge $t \in T_S$. Eine entsprechende Zuordnung von Bewertungsobjekten zu Vorgängen $t \in T_S$ kann auf Basis des Modells über die Identifikatoren der Vorgänge t_{guid} erfolgen.

4.4.3 Konsistenz zwischen Bauablaufplanung und Bauwerksplanung

Durch die Integration der Bauteile des BIM in die Bauablaufplanung ist eine Kontrolle des Bauablaufs hinsichtlich folgender Kriterien möglich:

Fehlende Zuordnung Durch die Differenz $B_{BIM} \setminus B_S$ erhält man die Menge $B_{notAssigned}$ der Bauelemente b, denen kein Vorgang zugeordnet wurde. Fehlende bauteilbezogene Vorgänge können dadurch ausgeschlossen werden.

Konsistenz - Reaktion auf Änderungen Die Generierung des Bauablaufs S basiert auf der vom Nutzer vorgegebenen Grobstrukturierung (siehe Kapitel 4.3.1), den Daten des BIM und dem Inhalt der Datenbasis T_{DB}. Tritt in einer der drei genannten Ausgangsinformationen eine Änderung auf, muss der Workflowgraph S neu generiert werden. Etwaige Änderungen in jeder der zugrunde liegenden Ausgangsinformationen fließen automatisch in die Neuberechnung des Workflowgraphen S ein und werden dadurch berücksichtigt.
Die bei derzeitigen Modellen problematische Nachführung des Bauablaufs bei Planungsänderungen kann dadurch entscheidend verbessert werden.

4.4.4 Wiederverwendbarkeit

Durch die Beschreibung eines Vorgangs mit seinen Voraussetzungen und seinem Resultat wird die Wiederverwendbarkeit von Vorgängen ermöglicht. Dies geschieht, indem der Prozess der Bauablaufplanung auf Basis des vorgestellten Modells formal abgebildet wird. Die Zuordnung von Vorgängen zu Bauteilen des BIM ist dadurch nicht an ein konkretes BIM gebunden.

4.4.5 4D-Anbindung

Die Bauteile des BIM sind zentraler Bestandteil des vorgestellten Modells und vollständig im Bauablauf S integriert. Eine Visualisierung des Bauablaufs in Form einer 4D-Planung ist nur möglich, wenn das verwendete BIM geometrische Daten der Bauteile enthält und deren rechnergestützte Verarbeitung möglich ist. Weiterhin muss das Modell für eine 4D-Visualisierung um weiterführende Daten ergänzt werden. Dies betrifft die zeitliche Bewertung aller Vorgänge $t \in T_s$ des Bauablaufs für die Visualisierung von Dauern und eine entsprechende Vorgabe für die Art des Vorgangs, um eine entsprechende Visualisierung zu realisieren. Beispielsweise müsste ein Bauteil im zeitlichen Kontext nach dessen Erstellung eingeblendet bzw. nach dessen Rückbau ausgeblendet werden. Ebenso ist eine farbige Darstellung der Bauteile bezogen auf den Fortschritt des Vorgangs zu einem bestimmten Zeitpunkt sinnvoll. Ein entsprechendes Konzept für eine 4D-Visualisierung wird in [Koschorrek u. a. 2008] erläutert.
Eine Zuordnung weiterführender Visualisierungsparameter und Zeiten zu Vorgängen kann auf Basis des entwickelten Modells über den eindeutigen Identifikator t_{guid} eines Vorgangs realisiert werden.

5 Pilotimplementierung

Dieses Kapitel behandelt die umgesetzte Pilotimplementierung des Forschungsansatzes. Die Umsetzung erfolgt auf Basis der Programmiersprache Java. Die entwickelten Klassen und Module werden auf Basis der UML dargestellt, um eine generelle Beschreibung zu gewährleisten.

5.1 Vorgehensweise und Wahl der Technologie

Für den Nachweis der Tragfähigkeit des vorgestellten Lösungsansatzes wurde eine Applikation entwickelt, die den vorgestellten Ansatz implementiert. Dabei wurde der softwaretechnischen Unterstützung des Anwenders eine außerordentliche Beachtung beigemessen. Hieraus ergaben sich vielfältige Anforderungen an die zu entwickelnde Applikation. Neben vielen Grundanforderungen, die an Softwareapplikationen im Allgemeinen gestellt werden, lassen sich zwei Hauptanforderungen im Hinblick auf die Handhabbarkeit des Lösungsansatzes herausstellen. Zum einen muss der Anwender in die Lage versetzt werden, Vorgänge mit deren Vorbedingungen und Resultaten effektiv zu definieren. Zum anderen muss eine Möglichkeit der Kontrolle generierter Bauablaufpläne gegeben sein. Um dies zu erreichen, ist eine bauwerkorientierte Visualisierung aller Phasen des Generierungsprozesses unabdingbar.
Der softwaretechnische Entwurf basiert auf der formalen Darstellung in Kapitel 3 und 4. Er wird unter Verwendung der Unified Modelling Language (UML) beschrieben.

5.1.1 Wahl der Softwaretechnologie

Für die softwaretechnische Modellierung und spätere prototypische Umsetzung wird die Programmiersprache *Java SE* (Standard Edition) in der Version 6 (siehe [Oracle 2010c]) gewählt. Die Wahl der Programmiersprache *Java* begründet sich in erster Linie in der weiten Verbreitung der Sprache innerhalb der Forschung und Lehre - nicht nur im Bereich der Bauinformatik.
Weiterhin zeichnet sich *Java* durch Objektorientierung, Einfachheit und Robustheit aus. *Java* gehört zur Gruppe der interpretierten Sprachen. Eine in *Java* implementierte Software wird demnach nicht in einem auf das Zielsystem[1] ausgerichteten Maschinencode, sondern in einen von Hardware und Betriebssystem unabhängigen Bytecode compiliert. Dieser Bytecode wird von der *Java Virtual Machine (JVM)* interpretiert, für das

[1] Unter einem Zielsystem wird in diesem Zusammenhang die Kombination aus der eingesetzten Hardware und dem Betriebssystem (auch OS - Operating System) verstanden.

entsprechende Zielsystem compiliert und optimiert, bevor sie innerhalb der *JVM* ausgeführt wird. Eine entsprechende *JVM* ist für nahezu alle Betriebssysteme verfügbar. Dadurch kann ein in *Java* Bytecode compiliertes Programm, zumindest theoretisch, ohne Anpassung auf jeder Plattform ausgeführt werden, für die eine *JVM* existiert. *Java* erreicht dabei eine ähnliche Ausführungsgeschwindigkeit, wie sie für *C++* oder *C#* zu erwarten ist.
Java bietet ebenfalls die für die prototypische Umsetzung notwendige Anbindung an 3D-Subsysteme, wie OpenGL[2] und *DirectX*[3]. Die dazu notwendigen Bibliotheken sind nicht im Standardumfang des *Java SDK* (Standard Development Kit) enthalten, sind jedoch frei verfügbar. Als etablierteste Pakete sind *JOGL* (Java OpenGL - derzeitig Version 1.1.1a, vgl. [JogAmp 2010]) und *Java3D* (derzeitige Version 1.5.2, vgl. [java.net 2010]) zu nennen. Während *JOGL* lediglich die direkte Verwendung der *OpenGL Version 2.0* in *Java* Applikationen ermöglicht, bietet *Java3D* die 3D-Grafikprogrammierung auf Objektebene an. Weiterhin kann bei Verwendung von *Java3D* je nach Ausführungsplattform die *DirectX*- oder *OpenGL*-Schnittstelle, basierend auf demselben Bytecode, angesprochen werden.
Beide Pakete sind als Open Source verfügbar und werden derzeit aktiv weiterentwickelt. Seit *Java Version 6* ist die *JOGL*-Technologie teilweise in die *Java SE* integriert worden und ermöglicht nun die völlig kompatible Verwendung von *JOGL*-Komponenten innerhalb von *Java Swing*[4]-basierten graphischen Nutzeroberflächen. Ein Großteil des *Java2D*-Paketes[5] basiert nun auf der Verwendung der 2D-Funktionalität von *OpenGL* über *JOGL*.
Trotz der offensichtlich sehr guten Integration von *JOGL* in *Java* wird für die spätere Umsetzung *Java3D* gewählt. Ausschlaggebend hierfür ist die zu Grunde liegende objektorientierte Modellstruktur von *Java3D* für die Implementierung der virtuellen dreidimensionalen Umgebung.

5.1.2 Wahl der Datenbanktechnologie

Für die Abbildung und spätere prototypische Umsetzung des Modells ist der Einsatz eines relationalen Datenbank-Managementsystems (DBMS) als zielführend anzusehen. Dies begründet sich darin, dass infolge des gewählten Lösungsansatzes eine gezielte Suche innerhalb der Datenbasis, basierend auf verschiedenen Informationen, durchgeführt werden muss.
Die Anbindung einer Datenbank ist unter Verwendung der gewählten Programmiersprache *Java* möglich. *Java* bietet hierzu die *JDBC*[6]-Schnittstelle. *JDBC Version 4.0* ist Teil der *Java SE Version 6* und bietet eine einheitliche, *SQL*[7] basierte Kommuni-

[2] engl.: Open Graphics Library - plattform- und sprachenunabhängige Schnittstelle für die Implementierung von Software mit 2D- und 3D-Grafik.
[3] Multimedia Interface der Firma Microsoft unter anderem für 3D-Grafik (Direct3D) auf Microsoft Windows Betriebssystemen
[4] Plattformunabhängige Technologie für graphische Komponenten wie Fenster, Knöpfe, Dialoge usw.
[5] Paket für erweiterte graphische 2D-Operationen - in der Java 6 SE enthalten
[6] Java Database Connectivity - Menge von Schnittstellen für den Zugriff auf relationaler Datenbanksysteme mittels Java
[7] Structured Query Language - Sprache für die Definition und Manipulation relationaler Datenbanken

kation mit einer Vielzahl von *DBMS* verschiedener Hersteller.
Das dieser Arbeit zugrunde liegende *DBMS* und somit dessen *SQL*-Sprachumfang ist *JavaDB*. Es handelt sich hierbei um das offiziell unterstützte Open Source *DBMS Apache Derby* (siehe [Apache 2010; Oracle 2010a]). *JavaDB* ist vollständig in *Java* umgesetzt und kann dadurch in vollem Umfang in eine *Java*-Applikation eingebettet werden. Die Verwendung weiterer Technologien außerhalb *Java* kann dadurch in diesem Fall vermieden werden. Dies trägt sowohl zur Durchgängigkeit und Verständlichkeit der angestrebten prototypischen Umsetzung als auch des Modellentwurfs bei.

5.1.3 Wahl des digitalen BIM

Aufgrund der infolge des entwickelten Modells bestehenden Anforderungen (vgl. Kapitel 4.2.1) und der Analyse verfügbarer BIM (vgl. Kapitel 2.1.1) wurden für die prototypische Umsetzung des entwickelten Konzepts die IFC zugrunde gelegt.

5.2 Anbindung des BIM auf Basis der IFC

Für die prototypische Umsetzung des vorgestellten Ansatzes auf Basis der IFC wird deren Version 2X3TC1 als BIM zugrunde gelegt. Dieses Kapitel beschreibt die Abbildung des IFC-Modells auf das innerhalb dieser Arbeit vorgestellte Modell. Dies betrifft die Überführung der im BIM enthaltenen Bauteile in die in Kapitel 5.3.1 vorgestellte Datenstruktur. Infolge der Komplexität der IFC werden nur die zum Verständnis der Umsetzung notwendigen Klassen und Attribute beschrieben. Weiterführende Informationen können in [Liebich 2009] und [buildingSmart 2010] nachgelesen werden.

5.2.1 Abbildung des Bauteiltyps und -identifikators

Alle in den IFC definierten Entitäten werden von der Basisklasse *IfcRoot* abgeleitet. Die Klasse *IfcRoot* definiert unter anderem das nicht optionale Attribut *GlobalId* vom Typ *IfcGloballyUniqueId*. Eine Instanz der Klasse *IfcGloballyUniqueId* repräsentiert einen *String* von 22 Zeichen Länge, der auch als Globally Unique Identifier (GUID)[8] bezeichnet wird und entspricht dem Zeichenumfang des Wertebereichs *ID* (siehe Kapitel 4.2.2). Abbildung 5.1 zeigt einen Ausschnitt der Vererbungshierarchie für die Klasse *IfcBuildingElement* und deren Kindklassen.
Für die Beschreibung von Bauteilen definieren die IFC insgesamt 20 Klassenbeschreibungen für Bauelemente. Diese sind im Einzelnen *IfcBuildingElementProxy, IfcCovering, IfcBeam, IfcColumn, IfcCurtainWall, IfcDoor, IfcMember, IfcRailing, IfcRamp, IfcRampFlight, IfcWall, IfcSlab, IfcStairFlight, IfcWindow, IfcStair, IfcRoof, IfcPile, IfcFooting, IfcBuildingElementComponent* und *IfcPlate*. Alle diese Klassen werden von der Basisklasse *IfcBuildingElement* abgeleitet, die, wenn man die Vererbungshierarchie bis zur Wurzel verfolgt, von der Klasse *IfcRoot* abgeleitet ist. Somit ist für alle mittels IFC instanziierten Bauteilobjekte eine eindeutige, persistente Identifikation durch das

[8] engl.: Globaler Eindeutiger Identifikator

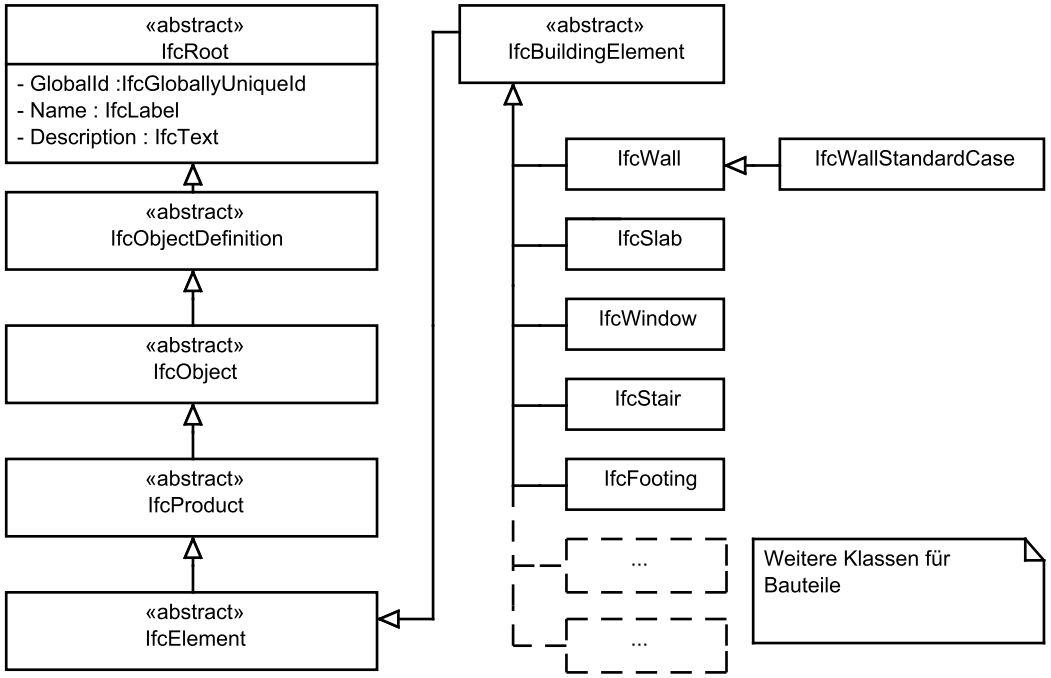


Abbildung 5.1: Vererbungshierarchie für Bauelemente in den IFC

Attribut *GlobalId* möglich. Der durch das Attribut gegebene *String* kann direkt als Wert a_v für das Attribut a_n =“id“ eines Bauteils $b \in B_{BIM}$ übernommen werden.
Der Typ eines Bauteils wird durch die innerhalb der IFC definierte Klassifizierung übernommen. Der Typ eines Bauteils entspricht dem Klassennamen einer Kindklasse von *IfcBuildingElement*. Für eine Instanz der Klasse *IfcWallStandardCase*, die eine konkretisierte Abbildung einer Wand darstellt, wird abweichend der Typ der Vaterklasse *IfcWall* übernommen. Das Attribut mit dem Namen a_n =“type“ erhält als Wert a_v demnach einen *String* mit dem Klassennamen der betrachteten Instanz, z.B. a_v =“IfcWindow“.
Weiterhin wird der Name des Bauteils, gegeben durch das Attribut *Name* der Klasse *IfcRoot*, für das Attribut mit dem Namen $a_n = „name$” übernommen.

5.2.2 Abbildung der Materialinformationen

Materialien können innerhalb der IFC über die objektivierte Relation *IfcRelAssociatesMaterial* jeder Instanz von *IfcRoot* und somit jedem *IfcBuildingElement* zugeordnet werden. Zugeordnet wird ein Objekt vom Typ *IfcMaterialSelect*, deren Kindklassen *IfcMaterial, IfcMaterialList, IfcMaterialLayerSetUsage, IfcMaterialLayerSet* und *IfcMaterialLayer* das zuzuordnende Material oder auch Materialschichten abbilden. Abbildung 5.2 zeigt die Vererbungshierarchie der genannten Klassen und die Verknüpfung von Materialien mit Bauteilen durch das Relationsobjekt *IfcRelAssociatesMaterial*.
Für die Abbildung der notwendigen Materialinformationen sind die folgenden Klassen von Bedeutung:

IfcMaterial: Die Klasse *IfcMaterial* bildet ein homogenes Material mit einer literalen Bezeichnung gegeben durch das Attribut *IfcMaterial.Name* ab.

IfcMaterialList: Die Klasse *IfcMaterialList* beschreibt einen abstrakten, mehrschichtigen Aufbau eines Materials durch die Angabe einer Liste von Materialien vom Typ *IfcMaterial*.

IfcMaterialLayer: Die Klasse *IfcMaterialLayer* definiert eine einzelne Schicht eines mehrschichtigen Materials. Objekte dieser Klasse werden gewöhnlich über eine Instanz von *IfcMaterialLayerSet* referenziert. Die Stärke der Materialschicht und ob es sich bei dieser Schicht um eine Luftschicht handelt, kann über die entsprechenden Attribute *IfcMaterialLayer.LayerThickness* bzw. *IfcMaterialLayer.IsVentilated* initialisiert werden.

IfcMaterialLayerSet: Die Klasse *IfcMaterialLayerSet* modelliert einen mehrschichtigen Materialaufbau aus einer beliebigen Anzahl von Schichten des Typs *IfcMaterialLayer*. Der Materialaufbau kann dabei optional durch das Attribut *IfcMaterialLayerSet.LayerSetName* benannt werden.

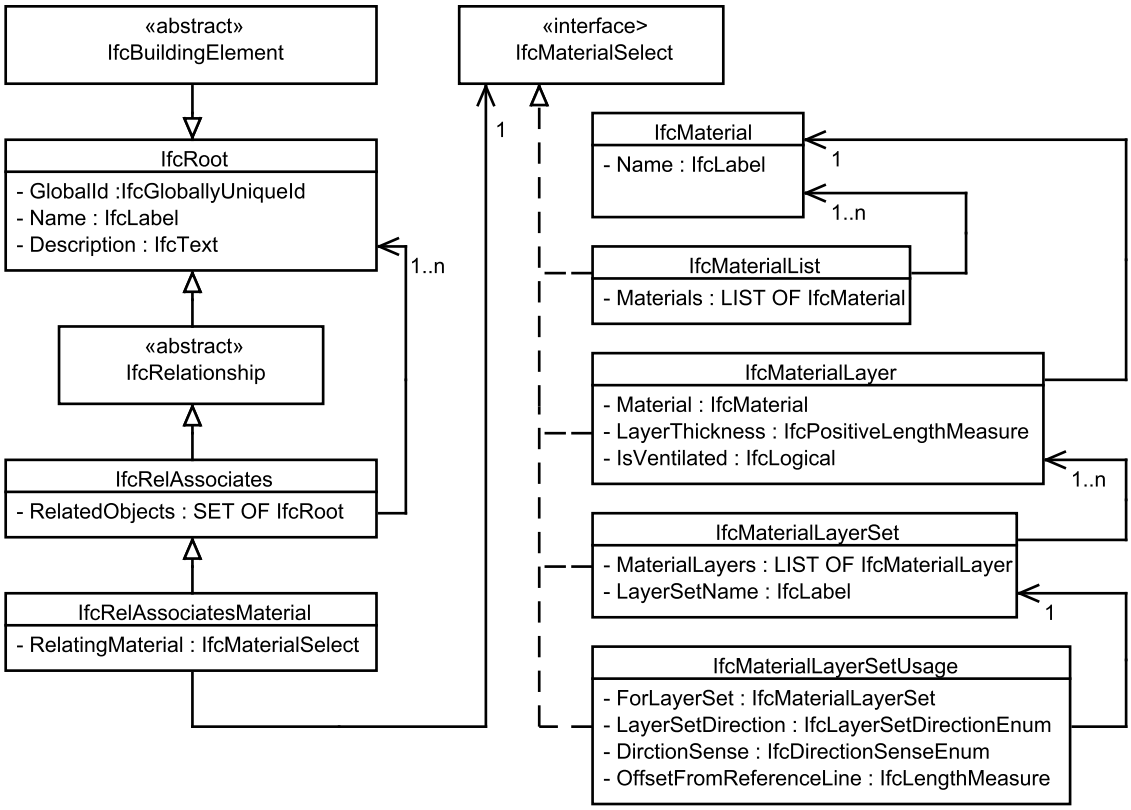


Abbildung 5.2: Vererbungshierarchie für materialbezogene Klassen in den IFC

Entsprechend dem entwickelten Modell werden alle Attribute a_{n_i} =“material.layer*.type“ mit dem Wert a_{v_i} =“IfcMaterial“ belegt. Alle Attribute a_{n_i} =“material.layer*.name“ werden mit dem Wert des Attributs *Name* der Klasse *IfcMaterial* initialisiert.

Besteht das einem Bauteil zugeordnete Material nur aus einem Objekt vom Typ *IfcMaterial*, steht * für den Index 1. Wurde dem Bauteil ein mehrschichtiger Materialaufbau zugeordnet, steht * für den jeweiligen Index der Materialschicht, bezogen auf den Index der jeweiligen Liste *IfcMaterialList.Materials* bzw. *IfcMaterialLayerSet.MaterialLayers*.

Beispiel: Nachfolgender Auszug aus einer IFC STEP Datei zeigt die Definition einer Wand mit einem mehrschichtigem Materialaufbau:

```
1 #3=IFCRELASSOCIATESMATERIAL('abcdefghijklmnopqrst02',#2,$,$
    ,(#22),#15);
2 #15=IFCMATERIALLAYERSET((#16,#17,#18),'Isolated outer wall');
3 #16=IFCMATERIALLAYER(#19,200.,$);
4 #17=IFCMATERIALLAYER(#20,80.,$);
5 #18=IFCMATERIALLAYER(#21,70.,$);
6 #19=IFCMATERIAL('Concrete');
7 #20=IFCMATERIAL('Mineral wool');
8 #21=IFCMATERIAL('Brick');
9 #22=IFCWALLSTANDARDCASE('295H6hv1z5T9dhGMC7AJX1',#2,'Wall 1',
    $,$,$,$,$);
```

Listing 5.1: Mehrschichtiger Materialaufbau für eine Wand nach [Liebich 2009]

Die Überführung der Daten erfolgt, ohne Angabe der ebenfalls geforderten Bounding-Box, bezogen auf das Bauteil und Material wie folgt:

$a_0 := (\mathbb{T}, "name", "Wall\ 1")$
$a_1 := (ID, "id", "295H6hv1z5T9dhGMC7AJX1")$
$a_2 := (\mathbb{T}, "type", "IfcWall")$
$a_3 := (\mathbb{T}, "material.layer1.name", "Concrete")$
$a_4 := (\mathbb{T}, "material.layer1.type", "IfcMaterial")$
$a_5 := (\mathbb{T}, "material.layer2.name", "Mineral\ wool")$
$a_6 := (\mathbb{T}, "material.layer2.type", "IfcMaterial")$
$a_7 := (\mathbb{T}, "material.layer3.name", "Brick")$
$a_8 := (\mathbb{T}, "material.layer3.type", "IfcMaterial")$

5.2.3 Abbildung der Bounding-Box

Die Bounding-Boxen der geometrischen Repräsentation von Bauelementen sind in den IFC nicht zwingend als Objekte des Modells verfügbar. Innerhalb der IFC kann ein Bauelement mittels verschiedener zweidimensionaler und dreidimensionaler geometrischer Repräsentationsarten beschrieben werden. Jedem Objekt vom Typ *IfcProdukt*, und somit auch jedem *IfcBuildingEelement* kann eine unbestimmte Anzahl verschiedener Repräsentationsarten über dessen Attribut *Representation* vom Typ *IfcProductRepresentation* zugeordnet werden. Die Klasse *IfcProductRepresentation* definiert hierzu die Liste *Representations*, welche die, einem *IfcProduct* zugehörigen, Repräsentationsarten

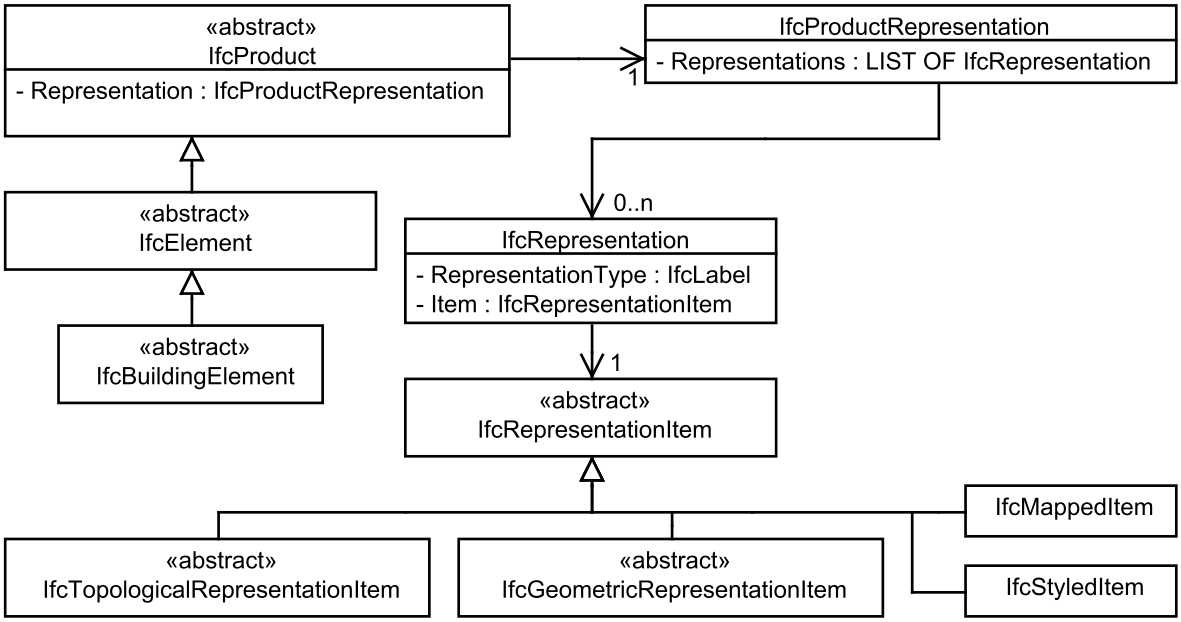


Abbildung 5.3: Modellierung von geometrischen Repräsentationen in den IFC

als *IfcRepresentation*-Objekte hält. Die Art der Repräsentation wird durch den String des Attributs *IfcRepresentaion.RepresentationType* näher spezifiziert. Abbildung 5.3 zeigt einen diesbezüglichen Auszug der IFC-Klassenstruktur. Beispiele für die Modellierung dreidimensionaler Repräsentationsarten sind *Brep*, *SweptSolid*, *CSG*, *Clipping* oder auch *BoundingBox*. Wobei ein *BoundingBox*-Objekt nach [Liebich 2009] nur als zusätzliche Repräsentationsart betrachtet wird, die definiert werden kann, um Applikationen, die nicht über entsprechende Interpretationsmöglichkeiten für komplexe Geometriemodelle verfügen, eine vereinfachte Darstellung zu ermöglichen.
Ist ein *IfcBoundingBox*-Objekt für ein Bauelement definiert, kann diese für die Bestimmung der Bounding-Boxen genutzt werden. Die Klasse *IfcBoundingBox* definiert dazu die Attribute *XDim*, *YDim*, *ZDim* sowie die Position der unteren linken Ecke *Corner*[9]. Diese Werte können direkt für die geforderten Attribute a_{n_i}=*boundingBox.xDim*, $a_{n_{i+1}}$=*boundingBox.zDim*, und $a_{n_{i+2}}$=*boundingBox.zDim* einer Bauteilbeschreibung *BuildingElement* übernommen werden. Für die geometrische Bestimmung der Voraussetzungen (siehe Kapitel 4.3.2) können die Eckpunkte der Bounding-Box durch Berücksichtigung der Position *Corner* berechnet werden.
Ist im BIM für ein Objekt *IfcBuildingElement* kein *IfcBoundingBox*-Objekt als Repräsentationsart definiert, muss für die Bestimmung der notwendigen Informationen die Geometrie gemäß der zugrunde liegenden Repräsentationsart interpretiert und die für die Abbildung der Bounding-Box geforderten Werte berechnet werden. Ein entsprechendes Werkzeug für die Interpretation komplexer Geometrien war zum Zeitpunkt der prototypischen Umsetzung nicht verfügbar und wurde aus diesem Grund im Team entwickelt (siehe Kapitel 5.4). Ein Beispiel für die Interpretation von *Brep* Repräsentationen der IFC ist in [Theiler u. a. 2009] dargestellt.

[9] entspricht dem Nullpunkt des lokalen Koordinatensystems der Bounding-Box im globalen Koordinatensystem

5.3 Softwaretechnisches Modell

Innerhalb des vorliegenden Kapitels erfolgt die Entwicklung eines softwaretechnischen Modells für den gewählten Lösungsansatz nach Kapitel 2 und 4.

5.3.1 Abbildung von Bauteilen

Für die softwaretechnische Abbildung von Bauteilen sind zwei Modelle notwendig. Einerseits wird die Abbildung der Bauteilobjekte für die Datenbasis, andererseits wird die Abbildung in Form von Klassen benötigt. Beide Abbildungen müssen in das jeweils andere Modell übertragbar sein.

Abbildung für die Datenbasis

Die Abbildung der Bauteilobjekte für die Datenbasis wird innerhalb einer Datenbank realisiert. Dies ermöglicht die effiziente Suche nach spezifischen Datensätzen und somit nach Bauteilbeschreibungen mit bestimmten Eigenschaften. Beispielsweise kann bei der Bestimmung von Vorgängen (vgl. Kapitel 4.3.2) direkt nach Vorgängen gesucht werden, deren Resultate einem bestimmten Bauteiltyp angehören. Die formale Abbildung lässt sich an dieser Stelle übertragen.
Abbildung 5.4 zeigt die graphische Darstellung des zugrunde liegenden ER-Modells[10] für die Abbildung der Bauteilobjekte innerhalb einer relationalen Datenbank.

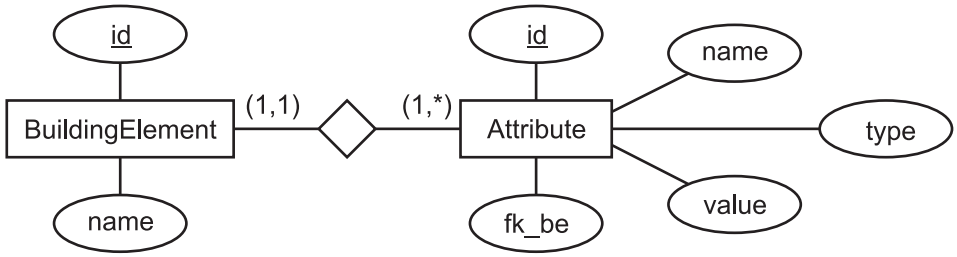


Abbildung 5.4: ER-Modell für die Abbildung von Bauteilen innerhalb einer relationalen Datenbank

Das Entity *BuildingElement* entspricht der formalen Abbildung der Menge der Bauelemente B. Das Attribut *id* dient hierbei als Primärschlüssel[11] für die eindeutige Identifikation des Datensatzes. Es entspricht nicht dem Attribut w_{id} der formalen Abbildung, welches innerhalb eines Datensatzes des Entity *Attribute* definiert wird (vgl. Tabelle 5.1).

[10] Entity Relationship Modell - Allgemeines Datenmodell für die Beschreibung von Entities und deren Beziehungen (vgl. [Claus u. Schwill 2003])
[11] Eindeutiger Identifikator eines Datensatzes

BuildingElement	Entity für die Abbildung eines Bauelements
id	Eindeutiger Identifikator des Datensatzes (Primärschlüssel)
name	Optionale Bezeichnung des Objekts als Text

Tabelle 5.1: Attribute des Entity *BuildingElement* nach Abbildung 5.4

Das Entity *Attribut* entspricht formal der Abbildung der Menge der Attribute A. Die Zugehörigkeit eines Attributs a_j zu einem Bauelement b_i wird über den Fremdschlüssel *Attribut.fk_be* realisiert (vgl. Tabelle 5.2). Ein Attribut a_j der Menge der Attribute A ist Attribut eines Bauelements b_i, wenn der Wert des Attributs *BuildingElement.id* mit dem Wert des Fremdschlüssels *Attribut.fk_be* identisch ist. Es gilt:

$$\begin{aligned} &Attribut \rightarrow A \\ &a_j \in b_i \Leftrightarrow Attribute.fk\_be = BuildingElement.id \end{aligned} \tag{5.1}$$

Attribute	Entity für die Abbildung eines Attributs
name	Name des Attributs als Text mit dem Wertebereich für a_n nach Formel 4.3
type	Typ des Attributs als textuelle Beschreibung für das eingesetzte Softwaresystem
value	Entspricht dem Wert des Attributs.
fk_be	Fremdschlüssel mit dem Verweis auf das Attribut ***BuildingElement.id***

Tabelle 5.2: Attribute des Entity *Attribute* nach Abbildung 5.4

Klassenbeschreibung

Ein Bauelement wird durch die in Abbildung 5.5 gezeigte Klasse *BuildingElement* abgebildet. Die Attribute der Klasse beschränken sich auf *name* für eine optionale Beschreibung des Bauelements.
Da im weiteren Verlauf der Modellentwicklung Bauelemente als Knoten eines Graphen anzusehen sind, wird das Interface *Node* eingeführt. Das Interface *Node* kennzeichnet die implementierenden Klassen für die Verwendung als Knotenobjekte von Graphen und vereinbart Methoden für das Hinterlegen von Attribut - Wertepaaren. Dies dient in erster Linie der Flexibilität des Modells. Es ermöglicht das Definieren zusätzlicher Informationen zur Laufzeit der Applikation. Im Hinblick auf die Ausrichtung der Arbeit sind beispielsweise Aufwandswerte für die Bewertung von Vorgängen zu nennen (vgl. Kapitel 4.2.4). Obwohl die Methode der Vererbung an dieser Stelle geeigneter erscheint, wird aufgrund der in *Java* fehlenden Merfachvererbung auf die Definition einer Schnittstelle zurückgegriffen.
Die konkrete Implementierung der Methoden des Interface *Node* wird innerhalb der Klasse *BuildingElement* gleichzeitig für die Abbildung der Attribute eines Bauelements

genutzt. Dazu wird für die Klasse *BuildingElement* das Attribut *properties* vom Typ *java.util.HashMap*[12] vereinbart. Die *java.util.HashMap properties* hält als *key* einen *java.lang.String*, welcher den Namen des Attributs analog zu a_n ausdrückt und als *value* die Instanz des jeweiligen Wertes. Die Instanz des Wertes ist vom Typ a_t bzw. *Attribute.type* des ER-Modells. Dabei muss der Datentyp a_t auf die verwendete Softwaretechnologie übertragbar sein. Die Methode *getProperty(String name)* liefert den Wert a_w des Attributs mit dem Namen a_n, der durch den übergebenen *String name* spezifiziert ist. Existiert kein Attribut mit dem Namen a_n, liefert die Methode als Ergebnis *null*. Durch Aufruf der Methode *setProperty(String name, Object object)* wird der Wert des Attributs a_i mit dem Namen a_n, gegeben durch den *String name* gesetzt. Existiert das Attribut mit dem Namen a_n noch nicht, wird es dem Objekt hinzugefügt. Die Methode *removeProperty(String name)* löscht das Attribut mit dem Namen a_n.

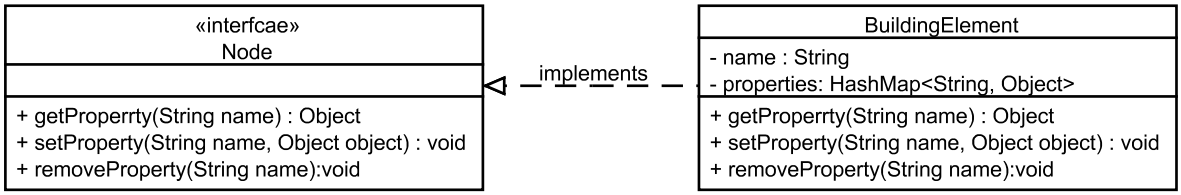


Abbildung 5.5: UML-Diagramm der Klasse *BuildingElement* und des Interface *Node*

5.3.2 Abbildung von Vorgängen

Die softwaretechnische Abbildung folgt auch an dieser Stelle der formalen Definition und ist sowohl für die Abbildung innerhalb der Datenbasis als auch in Form von Klassen notwendig.

Abbildung für die Datenbasis

Für die Abbildung von Vorgängen innerhalb der Datenbasis wird das Datenbankschema nach Abbildung 5.4 um die notwendigen Entities erweitert. Es handelt sich um die Entities *Task*, *Prerequisite* und *Result*. Das erweiterte ER-Modell ist in Abbildung 5.6 dargestellt.

Das Entity *Task* entspricht der formalen Abbildung der Menge der Vorgänge T. Für die eindeutige, persistente Identifikation eines Vorgangs wird das Attribut *guid* definiert (vgl. Tabelle 5.3). Der Wertebereich des Attributs ist definiert durch *ID* nach Formel 4.4. Somit gilt:

$$\begin{aligned} &Task \rightarrow T \\ &guid \rightarrow t_{guid} \end{aligned} \tag{5.2}$$

[12]Mapping zwischen einem Schlüsselwort (*key)* und einem Objekt (*object*) vgl. [Oracle 2010b]

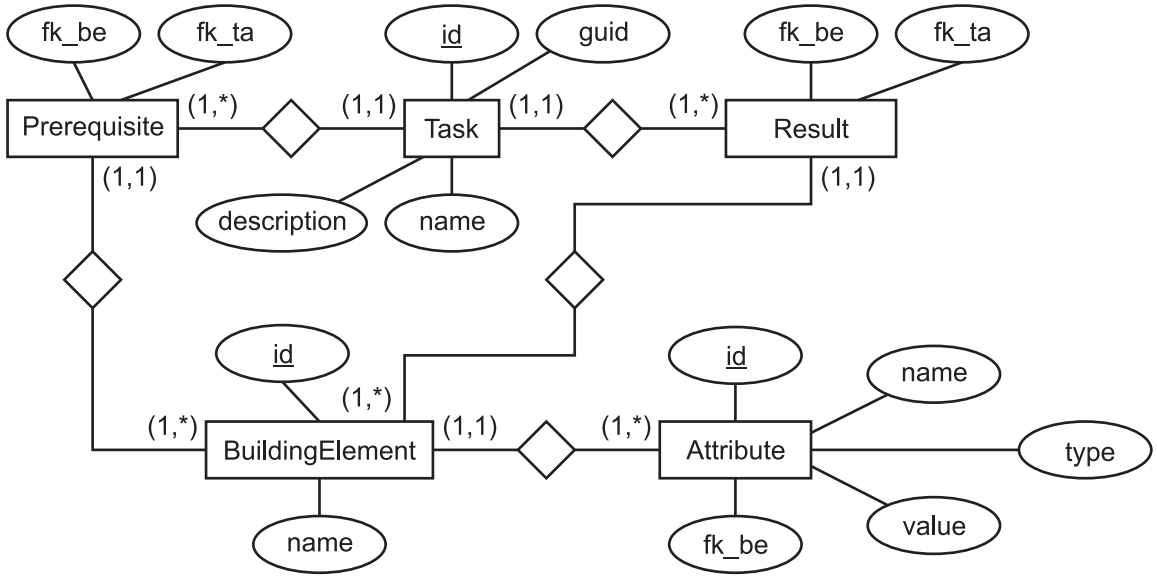


Abbildung 5.6: ER-Modell für die Abbildung von Vorgängen innerhalb einer relationalen Datenbank)

Task	Entity für die Abbildung eines Vorgangs
id	Eindeutiger Identifikator des Datensatzes (Primärschlüssel)
guid	Eindeutiger, persistenter Identifikator $\in ID$ nach Formel 4.5
name	Bezeichnung des Vorgangs als Text
description	Optionale Beschreibung des Vorgangs

Tabelle 5.3: Attribute des Entity *Task* nach Abbildung 5.6

Das Entity *Prerequisite* entspricht formal der Abbildung der Menge der Kanten P mit dem geordneten Paar $p(b,t) \in P$ und ist durch die Referenzierung der Fremdschlüssel *Prerequisite.fk_be* und *Prerequisite.fk_ta* gegeben (vgl. Tabelle 5.4). Fremdschlüssel *Prerequisite.fk_be* referenziert dabei einen Datensatz des Entity *BuildingElement* und somit ein Element b der Menge der Bauteile B. Fremdschlüssel *Prerequisite.fk_ta* referenziert einen Datensatz des Entity *Task* und demnach ein Element t der Menge der Vorgänge T. Es gilt:

$$\begin{aligned} &\mathit{Prerequisite} \rightarrow P \\ &p(\mathit{fk\_be}, \mathit{fk\_ta}) \rightarrow p(b,t) \end{aligned} \tag{5.3}$$

Prerequisite	Entity für die Abbildung einer Voraussetzung eines Vorgangs
fk_be	Fremdschlüssel mit Verweis auf auf das Attribut ***BuildingElement.id***
fk_ta	Fremdschlüssel mit Verweis auf auf das Attribut ***Task.id***

Tabelle 5.4: Attribute des Entity *Prerequisite* nach Abbildung 5.6

Das Entity *Result* entspricht der formalen Abbildung der Menge der Kanten R mit dem geordneten Paar $r(t, b) \in R$. Dieses ist durch die Referenzierung der Fremdschlüssel *Result.fk_ta* und *Result.fk_be* gegeben: $r(fk\_ta, fk\_be) \rightarrow r(t, b)$ (vgl. Tabelle 5.5). Fremdschlüssel *Result.fk_ta* referenziert dabei einen Datensatz des Entity *Task* und somit ein Element t der Menge der Vorgänge T. Fremdschlüssel *Result.fk_be* referenziert einen Datensatz des Entity *BuildingElement* und demnach ein Element b der Menge der Bauelemente B. Es gilt:

$$\begin{aligned} &Result \rightarrow R \\ &r(fk\_ta, fk\_be) \rightarrow r(t, b) \end{aligned} \tag{5.4}$$

Result	Entity für die Abbildung eines Resultats eines Vorgangs
fk_be	Fremdschlüssel mit Verweis auf das Attribut ***BuildingElement.id***
fk_ta	Fremdschlüssel mit Verweis auf das Attribut ***Task.id***

Tabelle 5.5: Attribute des Entity *Result* nach Abbildung 5.6

Klassenbeschreibung

Ein Vorgang wird durch die in Abbildung 5.7 gezeigte Klasse *Task* abgebildet. Die Klasse *Task* implementiert das Interface *Node* für die Verwendung ihrer Instanzen als Knotenobjekte eines Graphen. Die Attribute der Klasse bestehen aus der *guid* für einen eindeutigen, persistenten Identifikator t_{guid}, *name* für den Namen t_n des Vorgangs und *description* für die Beschreibung des Vorgangs t_d. Die Attribute *prerequisite* und *result* vom Typ *java.util.HashSet* stehen für die Menge der Vorbedingungen bzw. Resultate eines Vorgangs. Sie halten die Referenzen auf die jeweilige Instanz der Klasse *BuildingElement*.

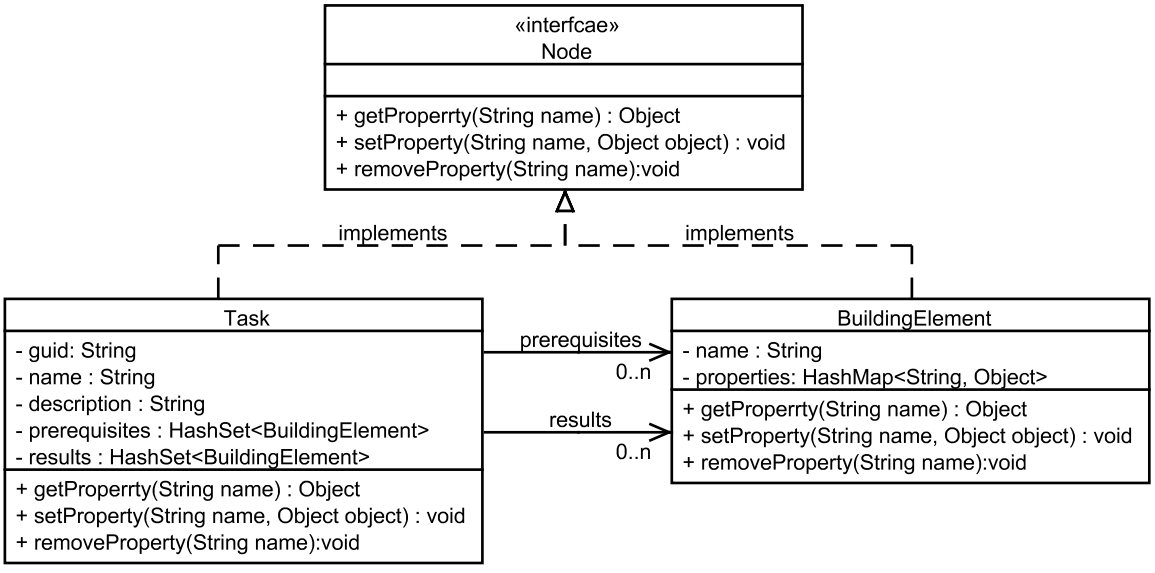


Abbildung 5.7: UML-Diagramm der Klassen *Task* und *BuildingElement*

5.3.3 Abbildung der Datenbasis

Die softwaretechnische Abbildung der Datenbasis erfolgt ausschließlich in Form einer relationalen Datenbank. Das bisher entwickelte Schema (vgl. Abbildung 5.7) stellt bereits alle notwendigen Entities dar. Als Datenbasis wird die Menge aller Vorgangsgraphen TG_{DB} betrachtet. Listing 5.2 bis 5.6 zeigen die SQL Anweisungen zum Erzeugen des Datenbankschemas.

```
CREATE TABLE BuildingElement
(id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  name VARCHAR(300));
```

Listing 5.2: SQL Query für das Erzeugen des Entity BuildingElement

```
CREATE TABLE Attribute
(id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
name VARCHAR(300) NOT NULL, type VARCHAR(300) NOT NULL,
value VARCHAR(300), fk_be INT);

ALTER TABLE Attribute ADD CONSTRAINT fk_be
FOREIGN KEY (fk_be) REFERENCES BuildingElement (id);
```

Listing 5.3: SQL Query für das Erzeugen des Entity Attribut

```
CREATE TABLE Task
(id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
guid VARCHAR(22) NOT NULL, name VARCHAR(300) NOT NULL,
description VARCHAR(30000));
```

Listing 5.4: SQL Query für das Erzeugen des Entity Task

```
CREATE TABLE Prerequisite (fk_be INT, fk_ta INT);

ALTER TABLE Prerequisite ADD CONSTRAINT fk_p_be
FOREIGN KEY (fk_be) REFERENCES BuildingElement (id);

ALTER TABLE Prerequisite ADD CONSTRAINT fk_p_ta
FOREIGN KEY (fk_ta) REFERENCES Task (id);
```

Listing 5.5: SQL Query für das Erzeugen des Entity Prerequisite

```
CREATE TABLE Result (fk_be INT, fk_ta INT);

ALTER TABLE Result ADD CONSTRAINT fk_r_be
FOREIGN KEY (fk_be) REFERENCES BuildingElement (id);

ALTER TABLE Result ADD CONSTRAINT fk_r_ta
FOREIGN KEY (fk_ta) REFERENCES Task (id);
```

Listing 5.6: SQL Query für das Erzeugen des Entity Result

Für die Anbindung des DBMS wurde die Serviceklasse *AliceDb* implementiert, die über entsprechende Methodenaufrufe das DBMS anspricht und basierend auf den Resultaten der SQL-Anfragen entsprechende Objekte instanziiert und zurückliefert. Abbildung 5.8 zeigt das UML-Diagramm der Klasse *AliceDb*.

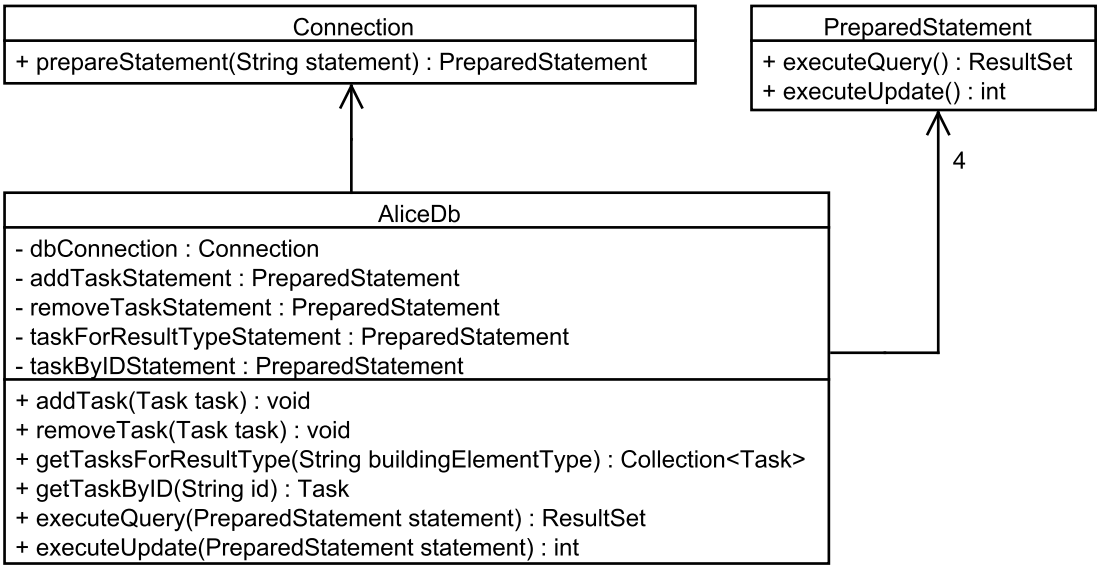


Abbildung 5.8: UML-Diagramm für die Klasse AliceDb

Klassenbeschreibung Das Interface *Connection* und die Klasse *PreparedStatement* entstammen dem package *java.sql*, das im Umfang des Java-SDK enthalten ist. Bevor ein DBMS über eine Instanz vom Typ *Connection* angesprochen werden kann, muss ein entsprechender Treiber geladen werden. Dies geschieht durch einfaches Instanziieren der entsprechenden Klasse. Für das eingesetzte DBMS *JavaDB* entspricht dies dem Aufruf *Class.forName("org.apache.derby.jdbc.EmbeddedDriver")*. Anschließend kann eine Instanz vom Typ *Connection* für das anzusprechende DBMS über den Aufruf der Methode *java.sql.DriverManager.getConnection(protocol + url + '';create=false'', properties) : Connection* erzeugt werden. Der Parameter *create* bewirkt, dass eine Datenbank mit der gegebenen URL gegebenenfalls erzeugt wird, sollte diese nicht existieren. In diesem Falle wird der Parameter mit dem Wert *false* belegt, was das Erzeugen verhindert. Das Objekt Properties ist vom Typ *java.sql.Properties* und übergibt über dessen Attribute einen Nutzernamen und Passwort für das DBMS. Über die vereinbarte Methode *prepareStatement* des Interface *Connection* werden die vordefinierten SQL Abfragen nach dem Schema *statement = connection.getConnection().prepareStatement ("SELECT * FROM TASK WHERE TASK_ID=?")* definiert. Der Platzhalter *?* wird dann für die konkrete Anfrage durch den Aufruf der Methode *statement.setString(1, id)* ersetzt (beispielsweise *id="123456789abcdefhjugd_e"*) und durch den Aufruf der Methode *ResultSet result = statement.execute()* ausgeführt.

5.3.4 Abbildung des Bauablaufs

Die softwaretechnische Abbildung erfolgt durch die in Abbildung 5.9 dargestellten Klassen. Die Klasse *BipartiteGraph* modelliert den bipartiten Graphen entsprechend der formalen Abbildung nach Kapitel 3.1. Das Attribut *nodesA* hält die Menge der Vorgänge T, *nodesB* die Menge der Bauelemente B, *edgesAB* die Menge der Resultate R und das Attribut *edgesBA* die Menge der Voraussetzungen P.
Die Kanten $r(t, b) \in R$ und $p(b, t) \in P$ werden mittels der Klasse *OrderedPair* als geordnetes Paar von *Node*-Objekten abgebildet. Die Mengen der Kanten werden intern durch eine Instanz der Klasse *java.util.TreeSet* verwaltet, um schnelle Mengenoperationen innerhalb des *java.util.TreeSet* zu ermöglichen.

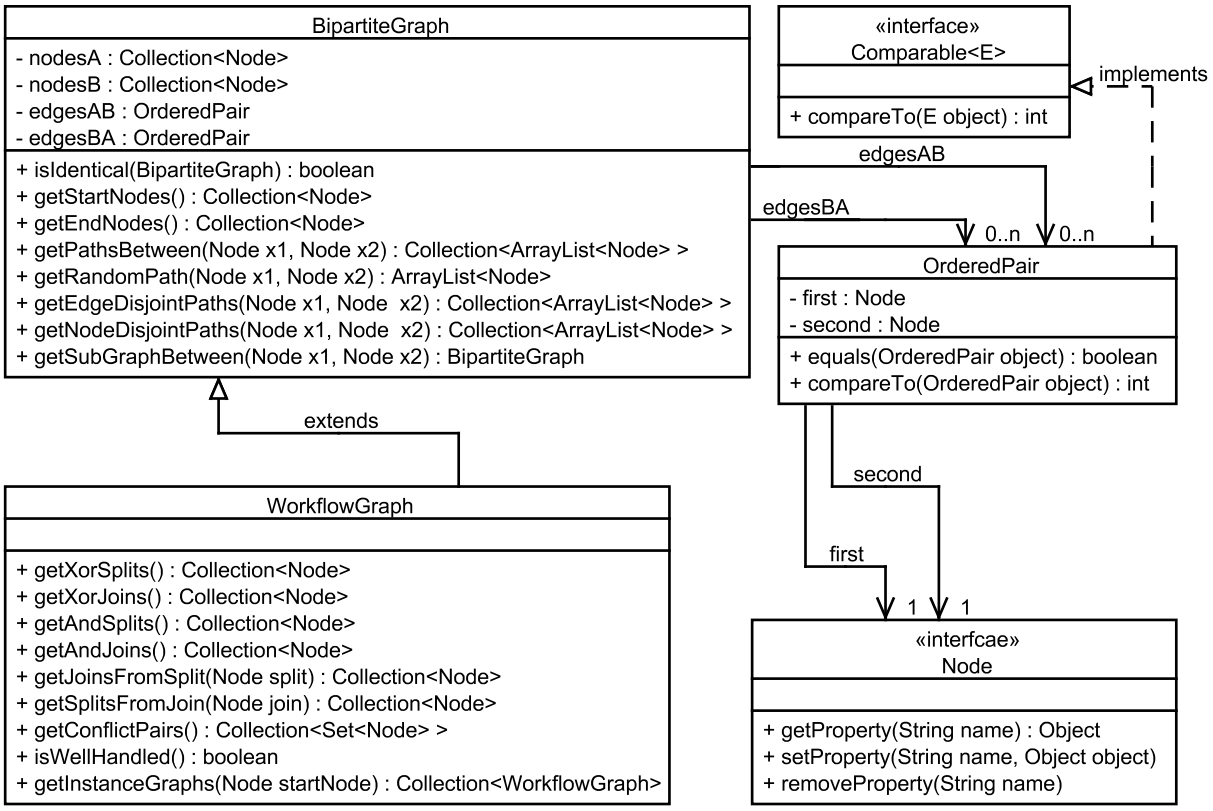


Abbildung 5.9: UML-Diagramm für die Klasse BipartiteGraph

Die im UML-Diagramm angegebenen Methoden der Klassen *BipartiteGraph* und *WorkflowGraph* entsprechen den in Kapitel 3 erläuterten Algorithmen.

5.3.5 Abbildung der Grobstrukturierung

Für die softwaretechnische Abbildung der Grobstrukturierung werden die Klassen *StructureUnit* und *StructureGraph* modelliert. Die Klasse *StructureGraph* entspricht dem schlichten Graphen $SG(SU, O)$ und enthält als Knoten Objekte vom Typ *StructureUnit*. Die Klasse *StructureUnit* entspricht hierbei der softwaretechnischen Abbildung einer

Struktureinheit *su*. Weitere Abhängigkeiten sind in Abbildung 5.10 als UML-Diagramm dargestellt.

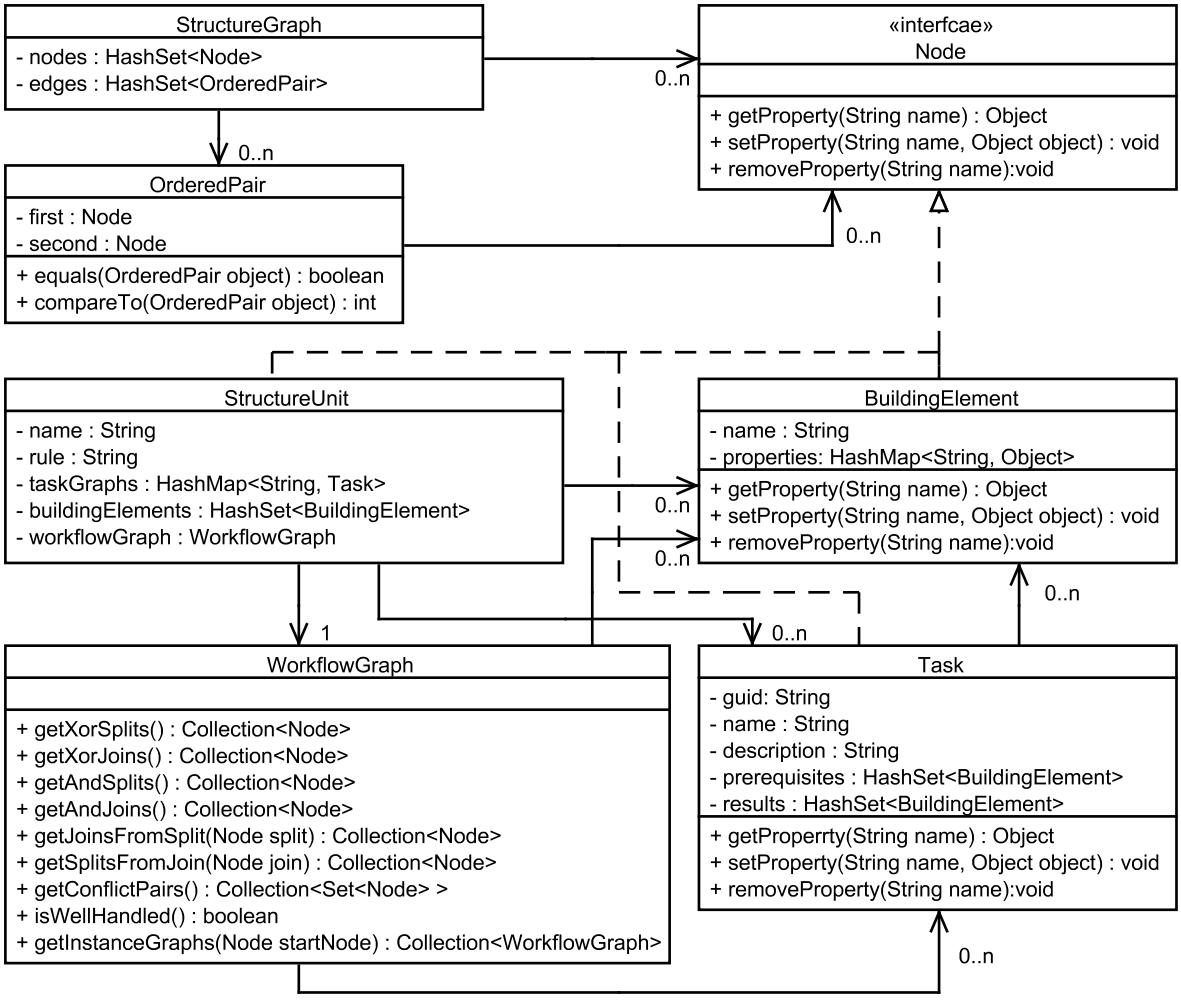


Abbildung 5.10: UML-Diagramm für die Klasse StructureUnit

5.4 Bestandteile und Interaktion der Softwarepakete des Prototypen

Die Bestandteile des Prototypen und deren Interaktion ist in Abbildung 5.11 dargestellt. Komponenten, die den Kern der vorliegenden Arbeit betreffen, werden mit ***Alice*** im Namen bezeichnet (**Al**ternatives **I**n **C**ivil **E**ngineering[13]). Die Softwarekomponenten *IFC STEP - PARSER / WRITER*, *IfcModel* und *3D / 4D Viewer* wurden im Team entwickelt. Eine Bereitstellung dieser Komponenten ist unter http://www.openifctools.org geplant und bereits zum Teil erfolgt. Die Funktionalität der verwendeten Komponenten wird nachfolgend erläutert.

[13]engl.: Alternativen im Bauingenieurwesen

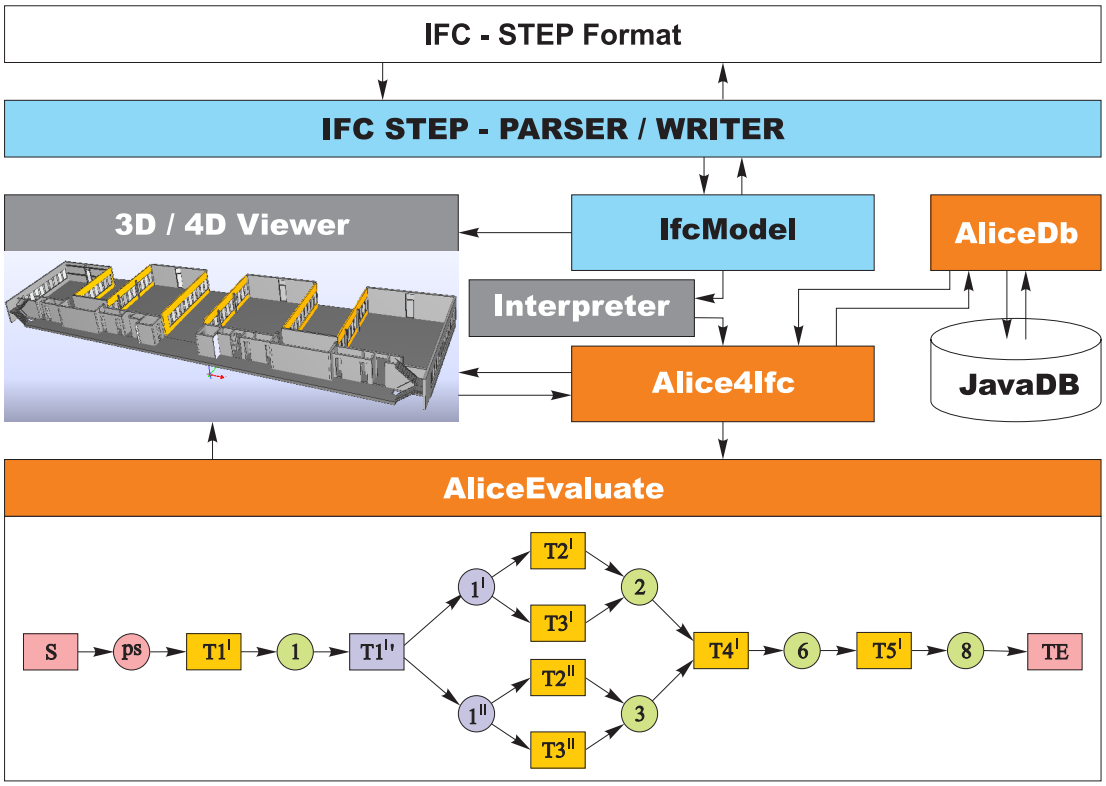


Abbildung 5.11: Bestandteile und Interaktion der Softwarepakete des Prototypen

5.4.1 IFC STEP - PARSER / WRITER und IfcModel

Für den notwendigen Zugriff auf IFC basierte BIM wurde im Team ein Softwarepaket entwickelt. Das Paket ermöglicht das Lesen und Schreiben IFC2x3TC1 basierter STEP-Dateien gemäß [ISO 2002a]. Das Paket bietet objektorientierten Zugriff auf die IFC-Objekte einer geparsten Datei. Die Umsetzung basiert auf dem Prinzip des *Early Binding*.

Zu diesem Zweck wurde unter Verwendung des Parser Compilers *JavaCC*[14] (siehe [Copeland 2007]) auf Basis des in EXPRESS (siehe [ISO 2004]) verfügbaren Schemas für die IFC Version 2x3TC1 ein limitierter[15] EXPRESS zu Java Compiler entwickelt. Dieser erzeugt die im IFC2X3TC1 Schema definierten Entities als Java-Klassen im *package ifc2x3tc1*. Abbildung 5.12 zeigt das UML-Diagramm.

[14] https://javacc.dev.java.net/, Abruf 17.11.2010
[15] Die Limitation besteht in dem Nichtbeachten der im Schema angegebenen Regeln und Funktionen.

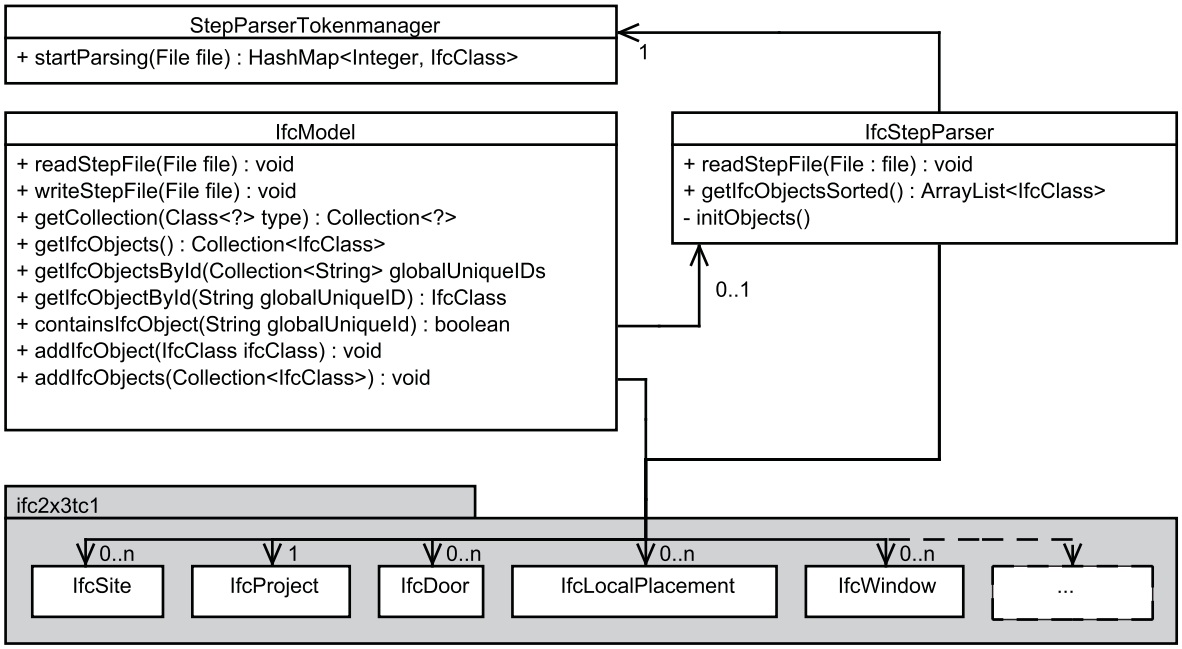


Abbildung 5.12: UML-Diagramm der Klasse *IfcModel*

Klassenbeschreibung

Der für das Einlesen der IFC STEP Dateien notwendige Parser *IfcStepParserTokenManager* wurde ebenfalls unter Verwendung von *JavaCC* auf Basis der STEP-Sprachdefinition entwickelt. Er liefert über die Methode *startParsing(File file)* die generischen Informationen über alle im STEP File enthaltenen Objekte. Innerhalb der Klasse *IfcStepParser* werden die entsprechenden Java-Klassen des durch den EXPRESS zu Java Compilers erzeugten *package ifc2x3tc1* instanziiert und initialisiert.
Die Klasse IfcModel stellt Schnittstellen für das Lesen und Schreiben von IFC STEP-Dateien dar. Über die Methoden *IfcModel.readStepFile* und *IfcModel.writeStepFile* kann diese Funktionalität angesprochen werden. Die im Falle des Einlesens einer IFC STEP-Datei instanziierten Objekte werden in der Klasse *IfcModel* vorgehalten und können über die dargestellten Methoden abgerufen werden. Beispielsweise liefert der Methodenaufruf *getCollection(IfcWindow.class)* die Menge aller im Modell enthaltenen Objekte vom Typ *IfcWindow*.

5.4.2 3D/4D-Viewer und Interpreter

Die Komponente 3D/4D-Viewer stellt die Hauptapplikation des Prototypen dar. Auch diese Komponente wurde im Team entwickelt und ermöglicht die Visualisierung des IFC-Modells. Weiterhin bietet sie eine Schnittstelle für die 4D-Animation eines Bauablaufs.
Die Komponente *Interpreter* ist die prototypische Umsetzung der Arbeit von [Tulke 2010]. Sie stellt die Schnittstelle für die in Kapitel 4.3.1 erläuterten Systemanfragen für die Grobstrukturierung des Bauablaufs dar. Die Komponente ist innerhalb des 3D/4D-Viewers integriert und kann entsprechend angesprochen werden.

5.4.3 Alice4Ifc und AliceEvaluate

Abbildung 5.13 zeigt das UML-Diagramm für die Klassen *Alice4Ifc* und *AliceEvaluate*. Bei *Alice4Ifc* handelt es sich um eine Serviceklasse. Sie implementiert die erforderlichen Algorithmen für die Überführung der für den Generierungsprozess notwendigen Informationen aus dem IFC-Modell in das Modell nach Kapitel 4. Die Klasse bietet als Schnittstelle die Methode *getBuildingElement(IfcBuildingElement ifcBE)*, die ein *IfcBuildingElement*-Objekt in ein Objekt vom Typ *BuildingElement* konvertiert.

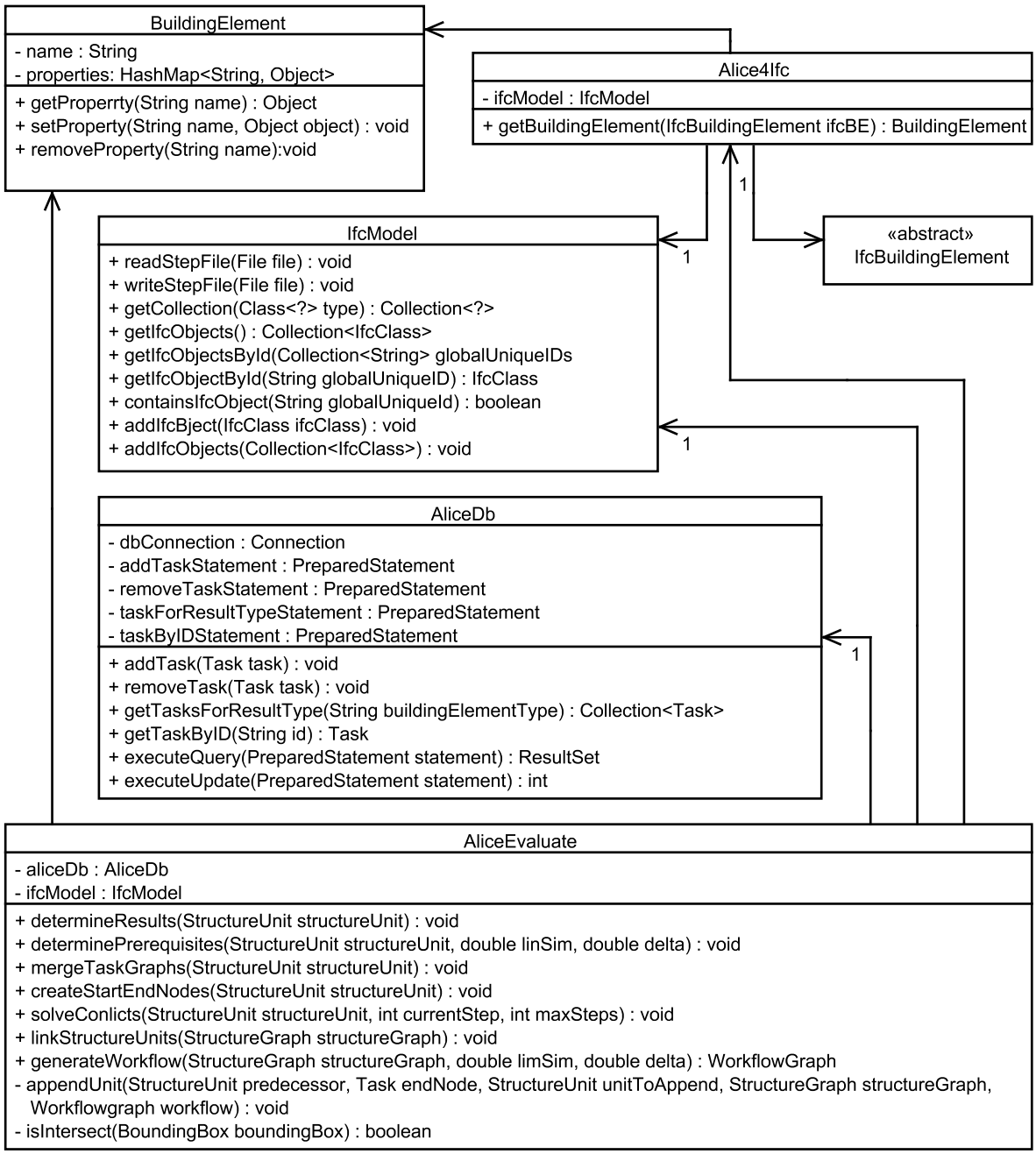


Abbildung 5.13: UML-Diagramm für die Serviceklasse *Alice4Ifc* und *AliceEvaluate*

Die Klasse *AliceEvaluate* beinhaltet die Algorithmen der Kapitel 4.3.2 bis 4.3.4 und stellt diese als Methoden bereit. Sie implementiert somit den Kern der Generierung.

5.4.4 Graphische Nutzeroberfläche

Abbildung 5.14 zeigt die graphische Nutzeroberfläche des Prototypen und einige seiner Komponenten. Alle Komponenten des Prototypen können innerhalb des Hauptfensters durch den Anwender frei angeordnet werden.

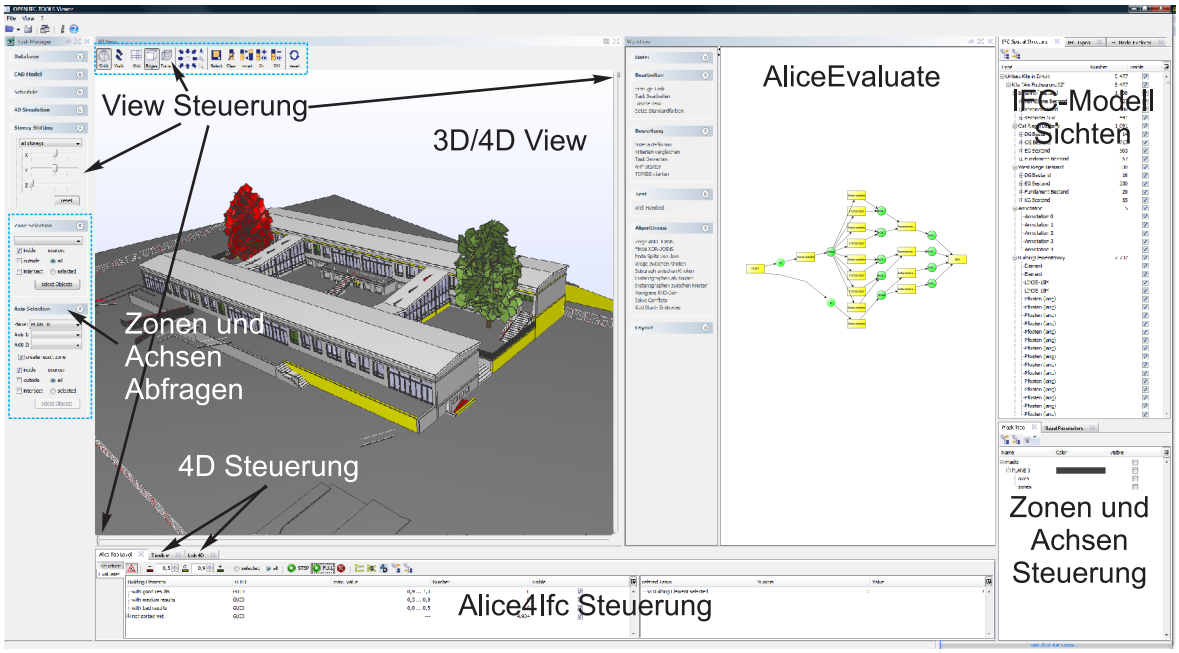


Abbildung 5.14: Graphische Nutzeroberfläche des Prototypen

3D/4D-View

Die Steuerung des 3D-Views ermöglicht das freie Drehen und Schieben des Modells. Weiterhin werden verschiedene Standardansichten auf das Modell angeboten und können durch den Anwender abgerufen werden.
Durch die beiden vertikalen und den horizontalen Schieberegler rechts bzw. unterhalb des 3D/4D-Views kann das Modell von unten, oben oder von vorn aufgeschnitten werden, um dem Anwender eine Sicht in das Bauwerksmodell zu ermöglichen.
Der Dialog *Storey Shifting* ermöglicht das Auseinanderschieben des Bauwerksmodells bezogen auf dessen Geschossebenen. Deren Lage kann durch Ziehen der Regler im Dialog *StoreyShifting* in jede Achsenrichtung beeinflusst werden. Die sich auf diese Weise ergebende Darstellung ist ähnlich einer Explosionsdarstellung und ermöglicht, entgegen dem erwähnten Aufschneiden des Modells, die gleichzeitige Sicht in alle Geschossebenen (siehe Abbildung 5.15). Weiterhin ist die Selektion von Bauteilen des BIM sowohl im 3D-View als auch in den IFC-Modell-Sichten möglich, was das Ein- und Ausblenden selektierter Bauteile ermöglicht.

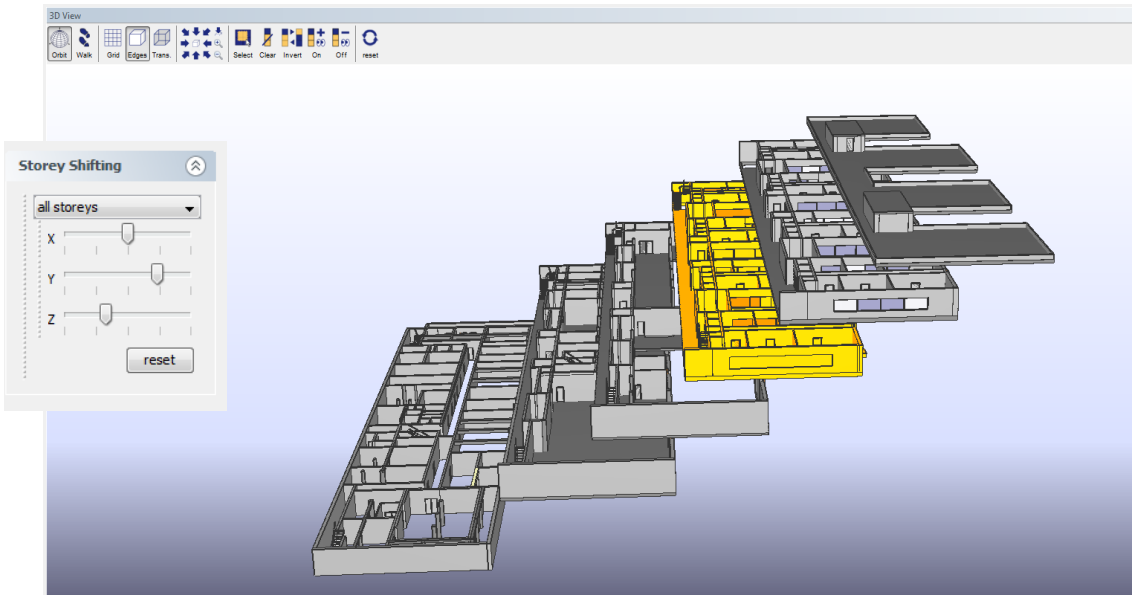


Abbildung 5.15: Beispiel für die Funktionalität *StoreyShifting*

Durch die Übertragung des generierten Workflowgraphen über die 4D-Schnittstelle des Viewers kann der sich daraus ergebene Arbeitsablauf animiert werden. Abbildung 5.16 zeigt die Darstellung während der Animation. In Konstruktion befindliche Bauteile werden zunächst eingeblendet und farbig markiert. Der zeitliche Ablauf kann durch den horizontalen Schieberegler der Registerkarte *Timeline* interaktiv beeinflusst werden. Die Farben der Markierung und die Art der Baumaßnahme kann über den Dialog *MaskView* gesteuert werden (vgl. [Koschorrek u. a. 2008]).

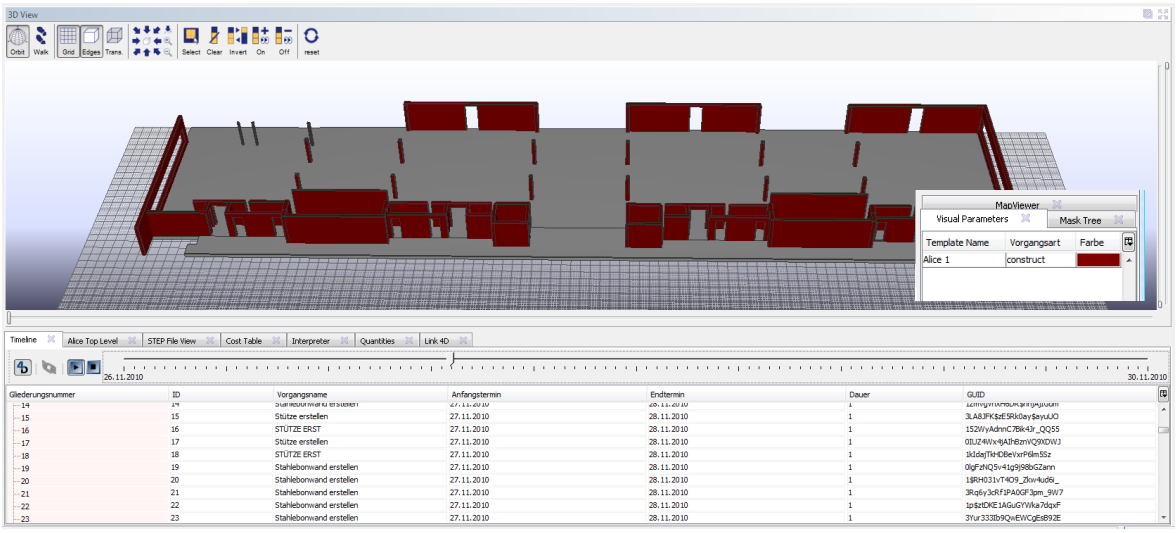


Abbildung 5.16: Beispiel für die 4D-Funktionalität des Viewers

Alice4Ifc-Steuerung

Die Komponente *Alice4Ifc-Steuerung* besteht aus zwei Registerkarten für die Steuerung der Grobstrukturierung und der Zuordnungs- und Generierungsphase.

Die Registerkarte *Structure* dient der Erstellung der Grobstrukturierung und ist in Abbildung 5.17 dargestellt. Die Modellierung der Grobstrukturierung erfolgt durch die Abbildung einer hierarchische Struktur für die zu modellierenden Struktureinheiten. Dabei ergibt sich die Definition des Strukturgraphen aufgrund der zugrunde liegenden Baumstruktur.

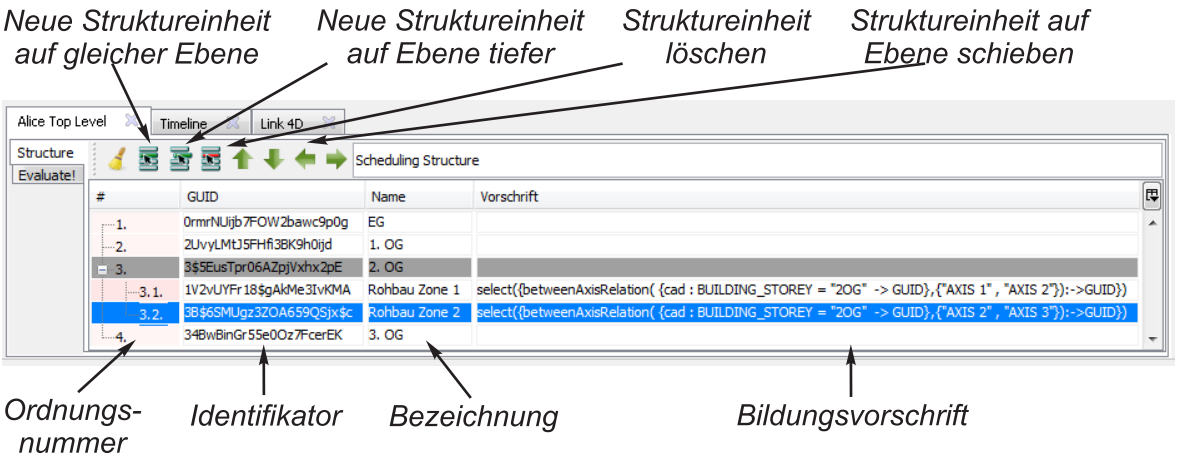


Abbildung 5.17: Registerkarte *Structure* für die Steuerung der Grobstrukturierung

Durch die Definition einer Bildungsvorschrift nach [Tulke 2010] wird für jede Struktureinheit deren zugehörige Bauteilmenge beschrieben. Dient eine Struktureinheit nur dem Zweck einer verständlichen Gliederung (z.B. EG, 1. OG, 2.OG und 3. OG), kann auf das Hinterlegen einer Bildungsvorschrift verzichtet werden. Ist für eine Struktureinheit keine Bildungsvorschrift hinterlegt, wird diese ausschließlich für die Kopplungsphase nach Kapitel 4.3.3 in Betracht gezogen.

Die in den Bildungsvorschriften verwendbaren Zonen und Achsen können mittels der Komponenten *Axis Selection*, *Zone Selection*, *Mask Tree* und dem *3D-View* vom Nutzer interaktiv definiert und überprüft werden. Das Erzeugen von Achsen oder Zonen ist jeweils auf eine durch den Nutzer zu bestimmende Ebene bezogen. Dabei ist die Anzahl der Ebenen nicht begrenzt. Nach Auswahl einer solchen Ebene kann auf deren Basis eine beliebige Anzahl von Achsen und Zonen definiert werden. Nach deren Definition sind diese innerhalb einer Abfrage nach [Tulke 2010] verfügbar. Abbildung 5.18 zeigt Beispiele für definierte Achsen und Zonen in einem Modell.

Die gelb markierten Bauelemente entsprechen jenen der ebenfalls dargestellten Achsenabfrage (*Axis Selection*). Es wurden alle Bauelemente selektiert, die sich innerhalb der durch den Zwischenbereich und der Höhe von Achse 2 und Achse 3 beschriebenen Zone befinden (vgl. [Tulke 2010]).

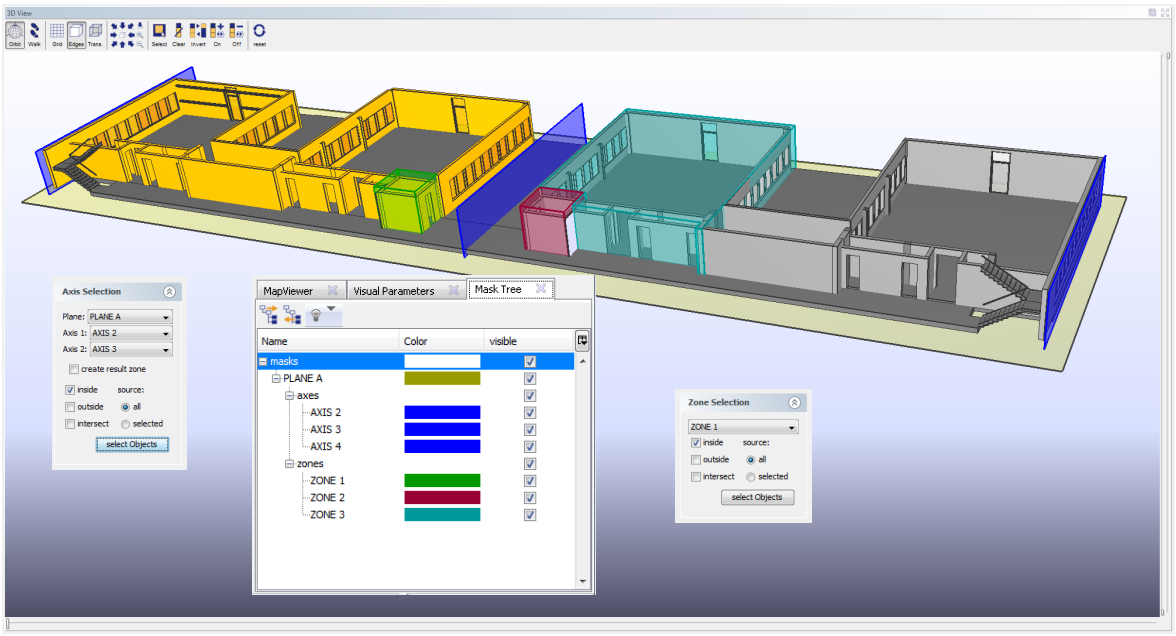


Abbildung 5.18: Definition von Achsen und Zonen im BIM

Die Registerkarte *Evaluate!* ermöglicht die Steuerung des Generierungsvorgangs. Die Registerkarte ist in Abbildung 5.19 dargestellt.
Für die Zuordnung von Vorgängen wurde ein Visualisierungsschema für alle Phasen der Generierung umgesetzt. Für die Zuordnung von Vorgängen zu Bauteilen des BIM wird eine Ampelfarbgebung verwendet. Dafür werden die zwei Grenzwerte $sim_{upper} = 0.9$ und $sim_{lower} = 0.5$ für die Gesamtähnlichkeit sim_o festgelegt. Bauteile, die als Resultat eines Vorgangs eine Ähnlichkeit $sim_o \geq sim_{upper}$ aufweisen, werden grün dargestellt, für $sim_{upper} \geq sim_o \geq sim_{lower}$ gelb und für $sim_o < sim_{lower}$ rot (die angegebenen Grenzwerte und Farben entsprechen den Standardvorgaben und können vom Anwender verändert werden).

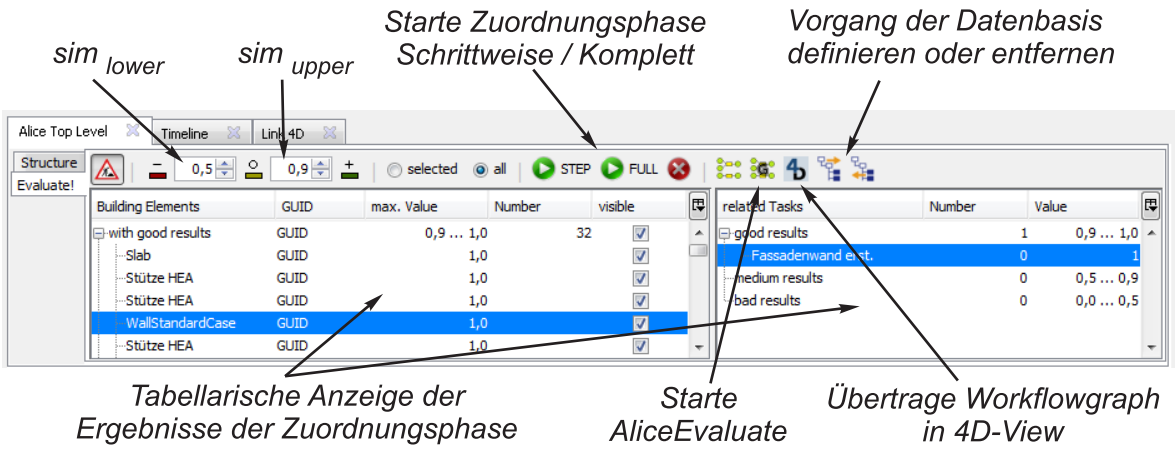


Abbildung 5.19: Registerkarte *Evaluate!* für die Steuerung der Generierung

Für die in der Standardeinstellung gelb und rot markierten Bauteile des BIM müssen demnach neue Vorgänge für die Datenbasis definiert werden oder können evtl. durch eine Anpassung vorhandener Vorgänge (gelb markiert) definiert werden. Abbildung 5.20 zeigt eine entsprechende Visualisierung.

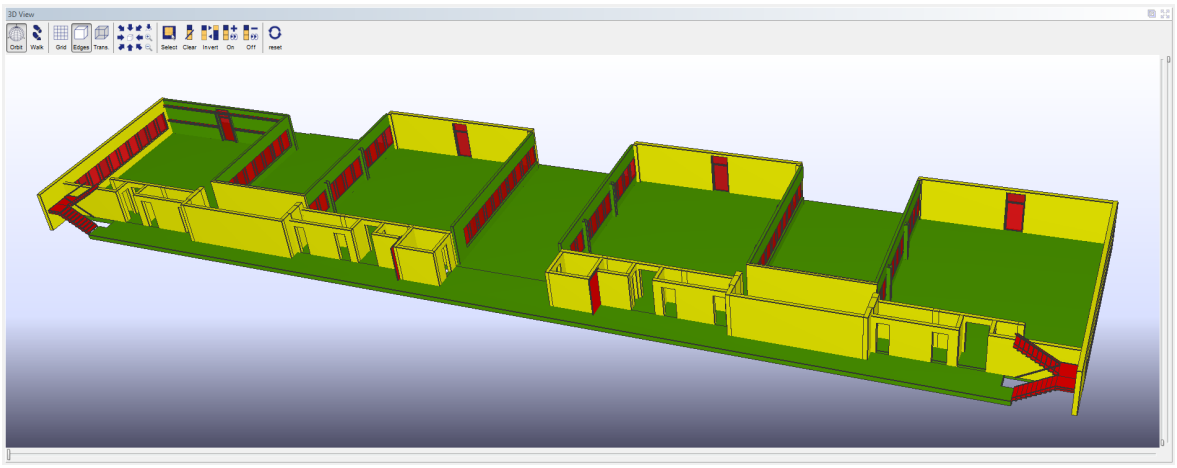


Abbildung 5.20: Beispiel für die Visualisierung der Zuordnung von Vorgängen in Ampelfarben

Die Definition von neuen Vorgängen für die Datenbasis wird über einen Dialog gesteuert. Dafür ist der Vorgang zu benennen und dessen Voraussetzungen und dessen Resultat zu definieren. Die Definition erfolgt durch die Selektion des Resultats bzw. der Voraussetzung des zu definierenden Vorgangs im 3D/4D-View und wird durch das Betätigen der jeweiligen Schaltfläche *Neu* übernommen.
Abbildung 5.21 zeigt das Erzeugen eines neuen Vorgangs mit der Bezeichnung „Fenster einbauen“. Die Definition des Vorgangs bezieht sich auf die blau markierte Wand mit dem darin befindlichen Fenster. Die Bauteilbeschreibung des Fensters wurde als Resultat, die blau markierte Wand als Voraussetzung des Vorgangs übernommen.

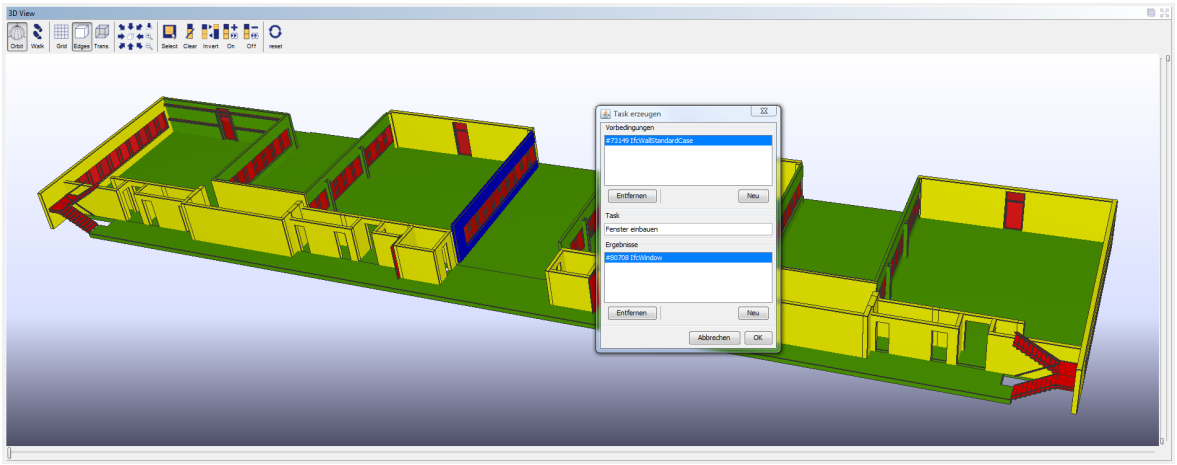


Abbildung 5.21: Definition eines neuen Vorgangs für die Datenbasis

AliceEvaluate

Die Komponente *AliceEvaluate* dient der Darstellung der generierten Workflowgraphen. Wurde sie vom Nutzer aktiviert, stellt sie nach erfolgter Generierung den Gesamtablauf oder den Ablauf für eine gewählte Struktureinheit und dessen Nachfolger im Strukturgraphen dar. Der Prototyp ermöglicht ebenfalls die schrittweise Ausführung der Konfliklösungs- und Kopplungsphase. Ebenso können die in den Kapiteln 3 und 4.3 vorgestellten Algorithmen auf den jeweils dargestellten Graphen angewendet werden.

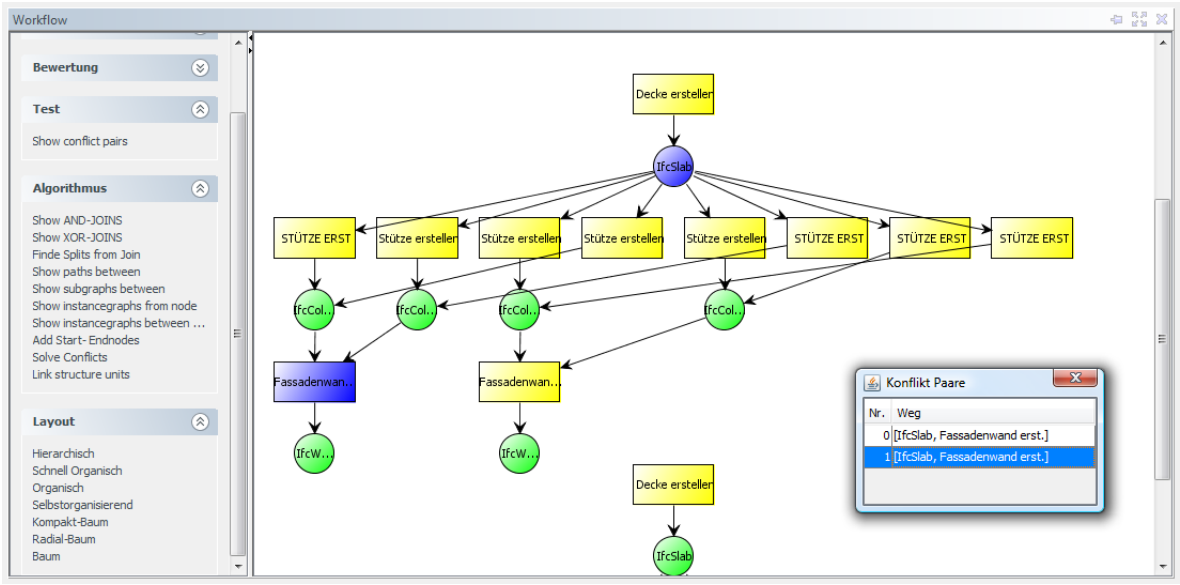


Abbildung 5.22: Komponente AliceEvaluate - Darstellung des bipartiten Graphen einer Struktureinheit vor der Konfliktlösungsphase

Abbildung 5.22 zeigt die Darstellung eines Graphen nach der Zuordnungsphase nach ausgeführtem *well handled* Test. Der dargestellte Dialog zeigt die gefundenen Konfliktpaare und visualisiert das in der Liste markierte Konfliktpaar im Graphen.

Die Selektion eines Bauteils des Graphen führt ebenfalls zur Selektion dieses Bauteils im 3D-View, was die Kontrolle des Graphen ermöglicht.

Auch innerhalb dieser Darstellung des generierten Workflowgraphen wird eine farbliche Kodierung verwendet, um den Anwender eine schnelle Übersicht zu ermöglichen (siehe Abbildung 5.23). Zugeordnete Bauteile des BIM werden als grüne Kreise dargestellt und Vorgänge als gelbes Rechteck. Pseudoknoten des Graphen erhalten die Farbe violett. Voraussetzungen eines Vorgangs, die nicht zugeordnet werden konnten, sind rot eingefärbt, um dem Anwender zu suggerieren, dass an dieser Stelle ein manueller Eingriff zur Problemlösung erforderlich ist (vgl. Kapitel 4.3.2).

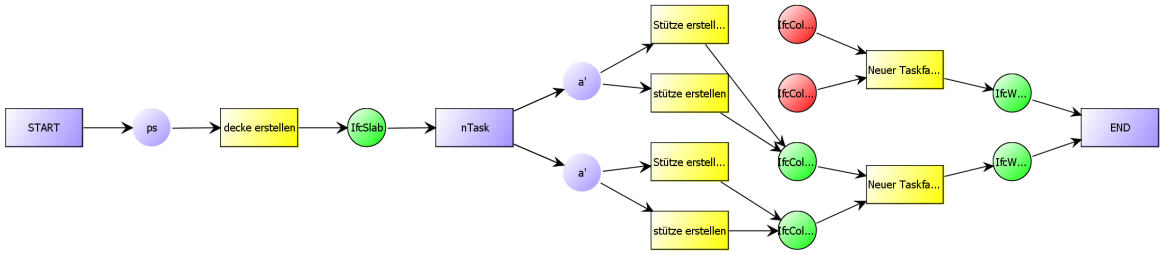


Abbildung 5.23: Farbkodierung der Darstellung des Workflowgraphen

5.4.5 Externe Softwarebibliotheken

Für die prototypische Umsetzung wurden neben dem Java SDK, JavaDB und Java3D weitere Bibliotheken genutzt. Die verwendeten Pakete sind nachfolgend unter Angabe ihres Verwendungszweckes gelistet:

lf2prod	Das Paket stellt die Swing-Komponenten *JTaskPane* und *JTaskGroup* zur Verfügung. Die Komponenten stellen kontextbezogene Kommandolisten zur Verfügung, die sowohl innerhalb des *3D/4D-Views* als auch für den *AliceEvaluate*-View verwendet wurden. Lizenz: Apache License, Version 2.0 Internet: http://www.l2fprod.com - Abruf: 24.11.2010
swingx	Aus dem Paket *swingx* wurde die Komponente *JXTreeTable* für die Darstellung der verschiedenen Sichten auf das IFC-Modell und die Alice4Ifc-Steuerung verwendet. Lizenz: Lesser General Public License (LGPL v. 2.1) Internet: https://swingx.dev.java.net - Abruf: 24.11.2010
jgraph	*jgraph* ist eine Java Swing kompatible Visualisierungsbibliothek für Graphen. Das Paket wird für die Darstellung des generierten Workflowgraphen innerhalb des *AliceEvaluate*-Views verwendet. Lizenz: open source (http://www.opensource.org) Internet: http://www.jgraph.com - Abruf: 24.11.2010
vldocking	Die Bibliothek bietet ein Java Swing Framework. Es ermöglicht die Implementierung von Anwendungen, mit durch den Anwender frei anpassbaren Applikationsfenstern. Innerhalb der prototypischen Umsetzung wurde die Bibliothek für die Integration der Komponenten der Hauptapplikation verwendet. Lizenz: Lesser General Public License (LGPL v. 2.1) Internet: http://code.google.com/p/vldocking - Abruf: 24.11.2010

6 Anwendungsbeispiel

In diesem Kapitel wird ein Beispiel auf Basis des entwickelten Prototypen dargestellt. Es beschreibt die Vorgehensweise für die teilautomatisierte Generierung eines Bauablaufplans anhand einer Struktureinheit aus der Sicht des Anwenders. Als Grundlage dient ein bereits realisiertes Bauprojekt, dessen zweidimensionale Zeichnungen in ein BIM auf Basis der IFC überführt wurden.

6.1 Bauwerksmodell - CIB-Weimar

Für die Darstellung der Funktionsweise des Prototypen dient das in Abbildung 6.1 gezeigte Gebäudemodell des CIB[1] Weimar. Es handelt sich um ein Forschungs- und Bürogebäude mit Versuchshallen sowie Lager- und Archivflächen.
Als Grundlage für das innerhalb dieses Kapitels vorgestellte Anwendungsbeispiel dient ein 3D-Gebäudemodell, das innerhalb eines studentischen Projektes unter Verwendung von Nemetschek Allplan 2008 anhand der 2D-Planungsunterlagen erstellt wurde (siehe [Ester u. a. 2008] und [Koschorrek u. a. 2008]).
Die für die innerhalb dieser Arbeit notwendigen Angaben für Materialien der Bauteile wurden entsprechend der Gebäudeplanung in das Modell integriert.

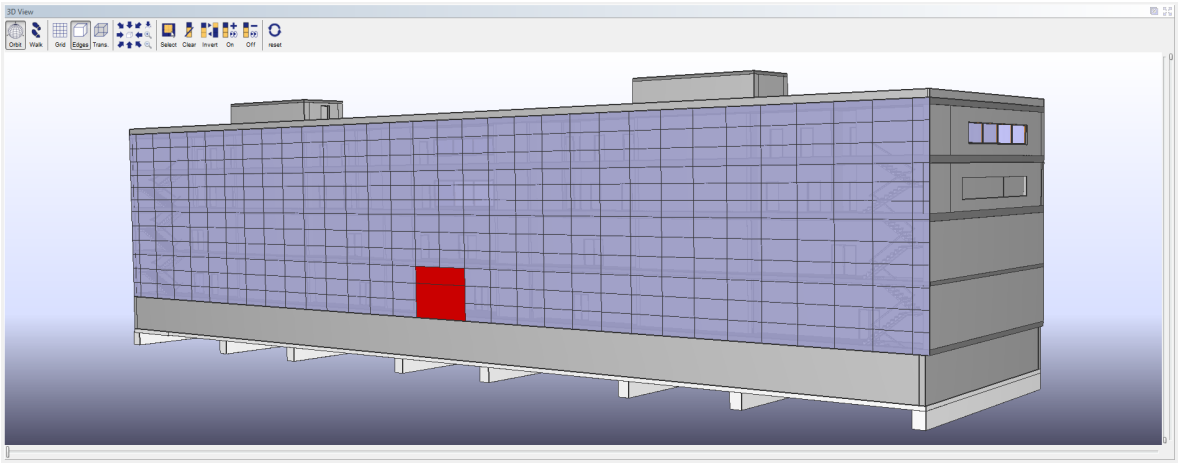


Abbildung 6.1: CIB Weimar - Bauwerksmodell des Anwendungsbeispiels im 3D/4D-View des Prototypen

[1] CIB Weimar „Centrum für Intelligentes Bauen“ - Technologiezentrum für Gründer, Unternehmer und Forscher des Bauwesens; http://www.cib-weimar.de/ - Abruf: 04.01.2011

6.2 Ausführung der Grobstrukturierung

Die Grobstrukturierung des BIM erfolgt innerhalb der bereits in Abbildung 5.17 dargestellten Registerkarte des Prototypen. Für die Darstellung des Beispiels wurde das in Abbildung 6.1 gezeigte Gebäude anhand der im BIM modellierten Geschossstruktur eingeteilt. Die Reihenfolge für die Erstellung der Geschosse wird dabei von unten nach oben beginnend mit der Erstellung des Fundaments vorgegeben. Die Auswahl basiert dabei auf der innerhalb des BIM enthaltenen Gebäudestruktur. Alle Bauelemente des zweiten Obergeschosses können beispielsweise durch die Bildungsvorschrift *select({cad:BUILDING_STOREY = "2. Obergeschoss"− > GUID})* gewählt werden. Durch Anwählen einer definierten Struktureinheit innerhalb der Baumansicht wird deren zugehörige Bildungsvorschrift ausgewertet. Das Ergebnis wird dem Nutzer durch die gelbe Einfärbung der betroffenen Bauelemente visualisiert (siehe Abbildung 6.3). Auf Basis dieser Funktionalität erhält der Anwender die Möglichkeit der visuellen Kontrolle bezogen auf Richtigkeit der von ihm hinterlegten Bildungsvorschriften.

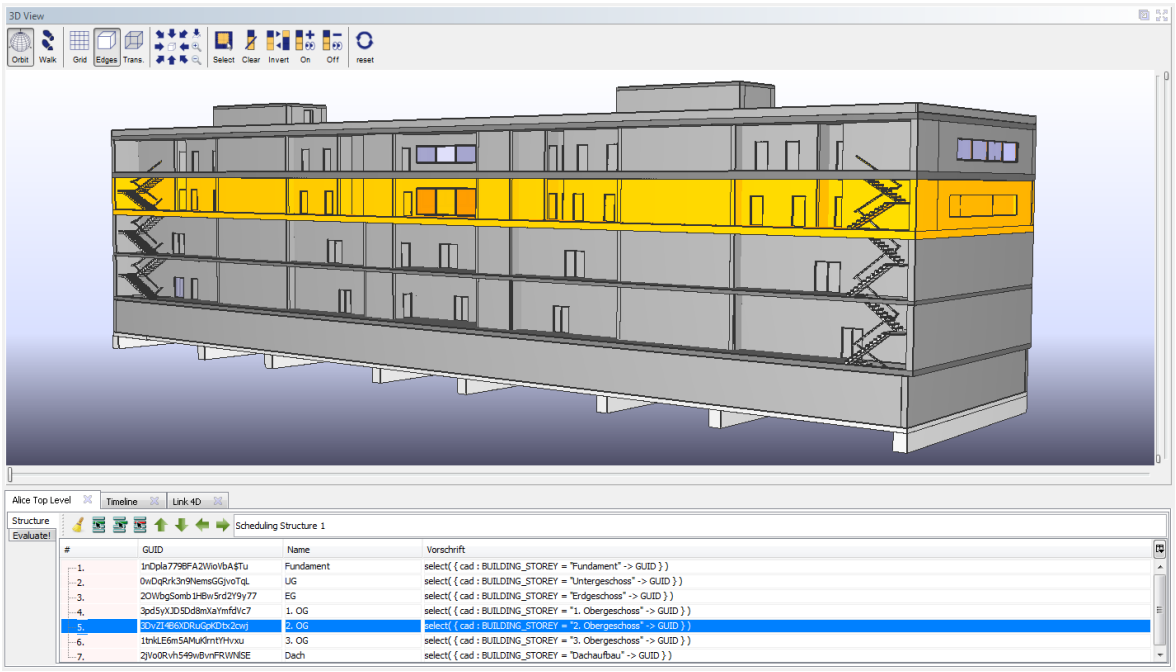


Abbildung 6.2: CIB Weimar - Darstellung des 2. OG im 3D/4D-Viewer

Um die Überschaubarkeit des Anwendungsbeispiels zu gewährleisten, wird die Menge der betrachtenden Bauteile auf die Bauteile des zweiten Stockwerks als Struktureinheit des Gesamtmodells eingeschränkt. Da die Algorithmen der Zuordnungs-, Generierungs- und Konfliktlösungsphase je Struktureinheit ausgeführt werden, stellt die ausschließliche Betrachtung dieser Bauelemente und deren weitere Unterteilung keine Einschränkung des Beispiels im Hinblick auf dessen Brauchbarkeit dar.
Auf die Darstellung der Kopplungsphase wird für das Beispiel verzichtet, da dieser Vorgang bereits ausführlich in Kapitel 4.3.3 dargestellt wurde und an dieser Stelle keine neuen Erkenntnisse bringt.

Die für das Beispiel betrachteten Bauteile für die Struktureinheit des zweiten Obergeschosses sind in Abbildung 6.3 dargestellt.

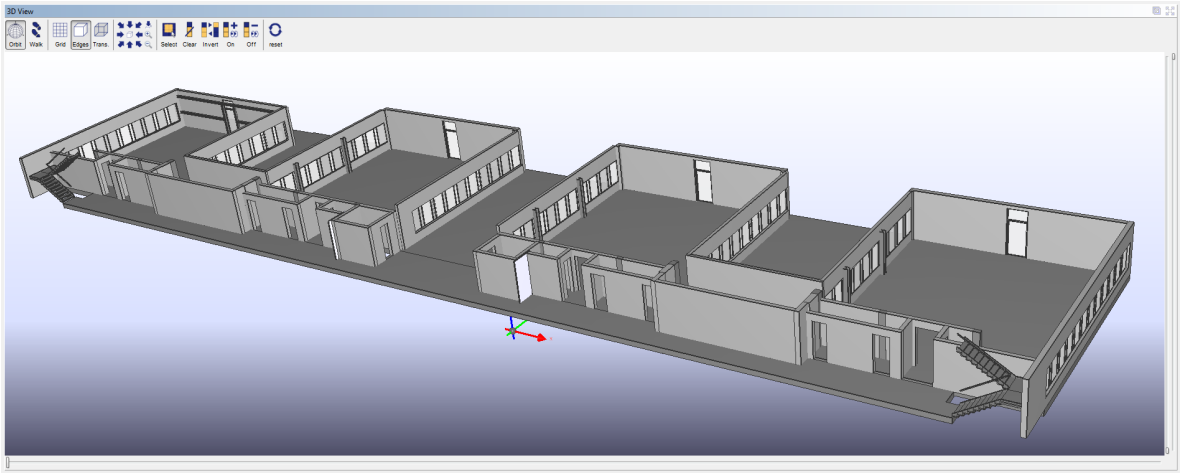


Abbildung 6.3: CIB Weimar - Darstellung des 2. OG im 3D/4D-Viewer

6.3 Ausführung der Zuordnungs-, Generierungs- und Konfliktlösungsphase

Nachfolgend werden die verschiedenen Phasen der Zuordnungs-, Generierungs- und Konfliktlösungsphase dargestellt.

Die innerhalb dieses Kapitels angegebenen Laufzeitangaben entsprechen dem Durchschnitt von zehn Messungen und beziehen sich auf die Ausführung des Prototypen auf einem Rechner mit Intel® Core™ 2 Quad CPU, 2.4GHz, Microsoft Vista 64 Bit-Betriebssystem mit 8 GB Arbeitsspeicher und der Java Virtual Machine Version 1.6.0_13. Die angegebenen Laufzeiten wurden jeweils bis zur vollständigen Visualisierung der jeweiligen Ergebnisse gemessen.

6.3.1 Umsetzung der Zuordnungsphase

Für das Beispiel wird angenommen, dass sich zu Beginn der Zuordnungsphase keine Vorgänge innerhalb der Datenbasis befinden. Bei einer leeren Datenbasis ergibt sich folgerichtig das in Abbildung 6.4 dargestellte Ergebnis nach Ausführung der Zuordnungsphase. Da den Bauteilen keine Vorgänge zugeordnet werden können, werden alle Bauteile rot markiert, um dem Anwender zu vermitteln, dass er für diese Bauteile Vorgänge definieren muss.

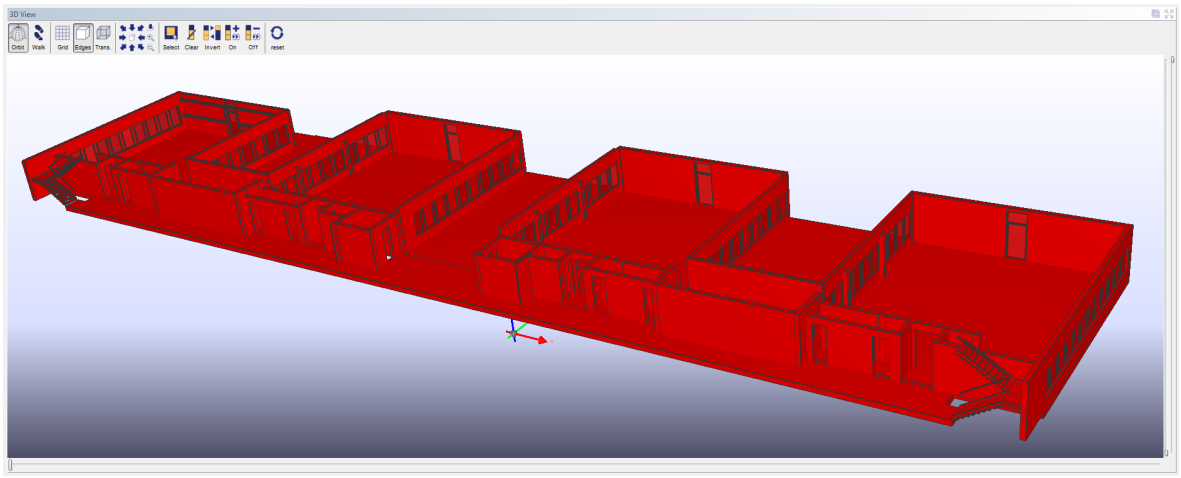


Abbildung 6.4: Ergebnis der Zuordnungsphase bei leerer Datenbasis

Ein Vorgang wird durch den in Abbildung 6.5 dargestellten Dialog vorgenommen (siehe auch Kapitel 5.4.4). Die Abbildung zeigt die Definition eines Vorgangs zum Erstellen einer Decke. Das betreffende Bauteil wird durch Anklicken selektiert, was dem Anwender durch eine blaue Einfärbung des Bauteils visualisiert wird. Durch das Betätigen des Dialogknopfes *Neu* des Dialogabschnittes *Ergebnisse* wird dessen Bauteilbeschreibung für die Definition des Vorgangs übernommen. Nach Benennung des Vorgangs und der Bestätigung des Dialogs mit *OK* wird dieser in die Datenbasis eingefügt. Ab diesem Zeitpunkt steht er für alle nachfolgenden Zuordnungsphasen zur Verfügung.

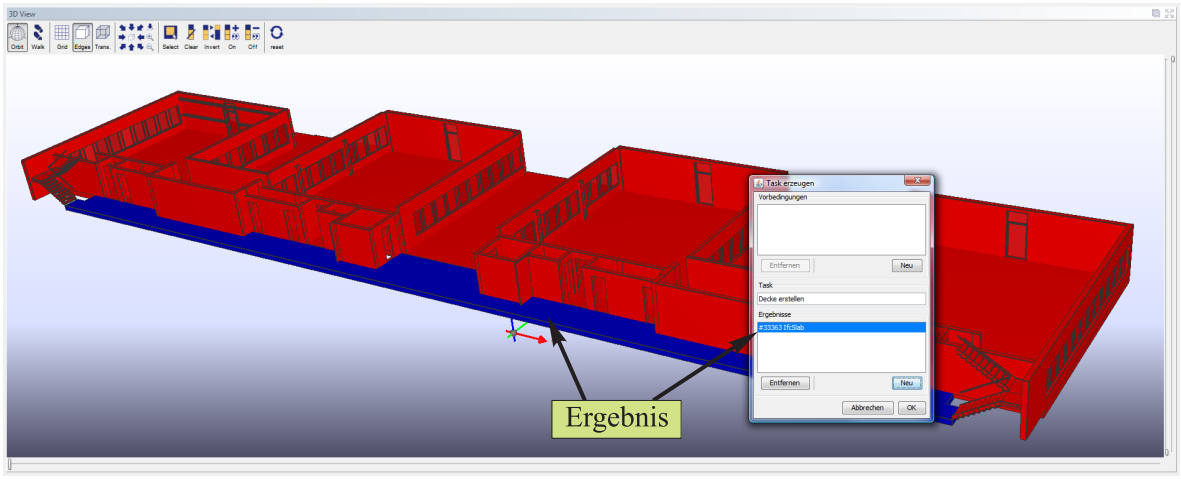


Abbildung 6.5: Definition eines Vorgangs für die Erstellung einer Decke

Abbildung 6.6 zeigt das Ergebnis der Zuordnungsphase nach der Definition des Vorgangs zum Erstellen einer Decke.

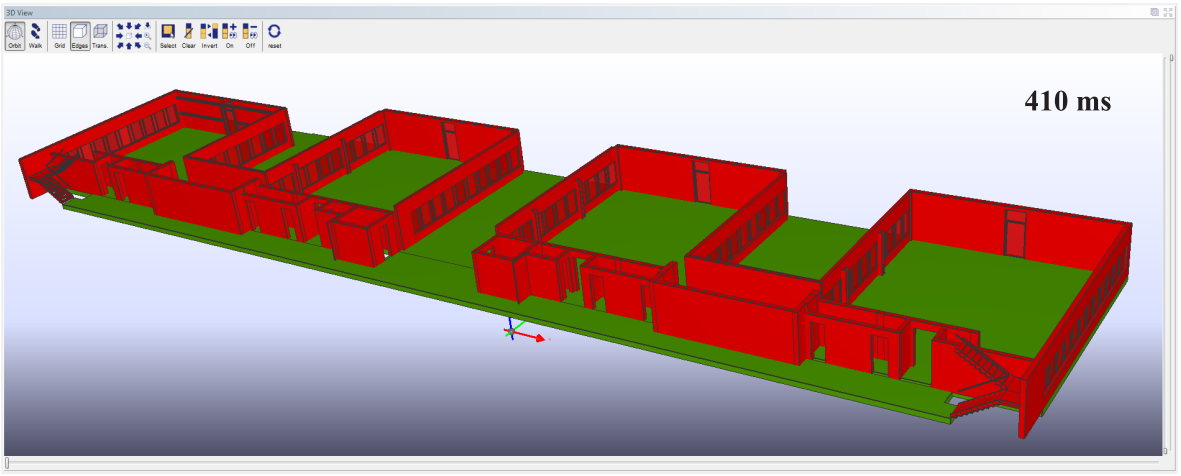


Abbildung 6.6: Ergebnis der Zuordnungsphase nach Definition des Vorgangs für die Erstellung einer Decke - Ergebnis nach 410 ms

Den zwei Deckenabschnitten der Struktureinheit wurde der neu definierte Vorgang zugeordnet, was dem Anwender durch die grüne Einfärbung signalisiert wird. Das berechnete Ähnlichkeitsmaß des Deckenabschnitts, dessen Bauteilbeschreibung im Dialog für das Ergebnis des Vorgangs übernommen wurde, beträgt $sim_o = 1,0$, das des zweiten Deckenabschnitts ergibt sich zu $sim_o = 0,97$ (Siehe Kapitel 4.2.3, Formel 4.15). Das hohe Ähnlichkeitsmaß sim_o von 0,97 ergibt sich infolge der sehr geringen Unterschiede beider Deckenabschnitte. Im Beispiel unterscheiden sich diese lediglich aufgrund ihrer Geometrie für die Ausdehnung der Bounding-Box in y-Richtung.
Abbildung 6.7 zeigt die Definition von zwei weiteren Vorgängen für die Errichtung einer Stahlstütze. Es handelt sich hierbei um verschiedene Vorgänge, die jeweils als Voraussetzung eine Decke benötigen und zum selben Resultat führen. Die definierten Vorgänge stellen demnach zwei mögliche Ausführungsvarianten für das Errichten einer Stahlstütze dar.

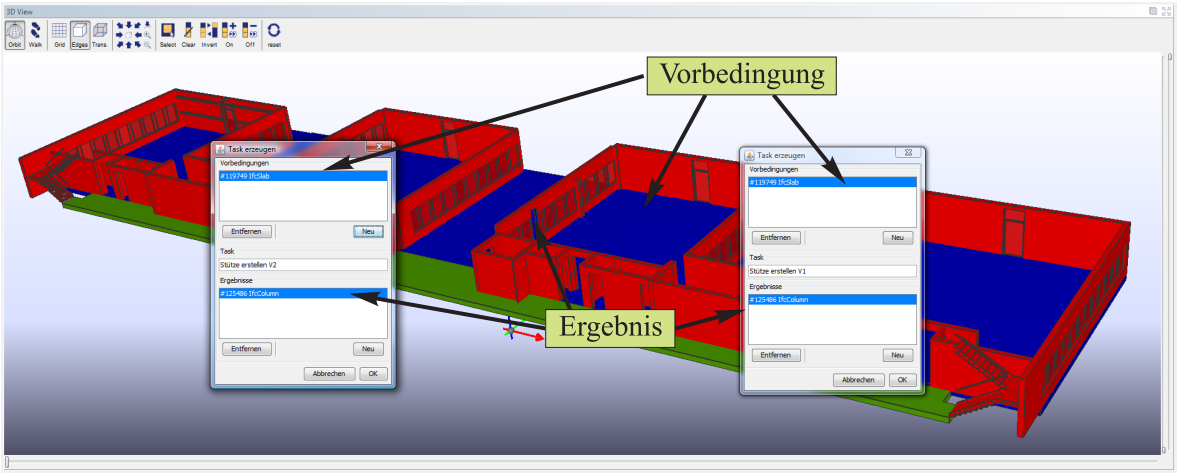


Abbildung 6.7: Definition zweier Vorgänge für den Einbau einer Stütze

Abbildung 6.8 zeigt das Ergebnis nach erneuter Ausführung der Zuordnungsphase. Allen Stützen der Struktureinheit mit einem Ähnlichkeitsmaß größer 0,9 wurden die beiden neu definierten Vorgänge zugeordnet. Das Ergebnis der Zuordnung wird ebenfalls tabellarisch innerhalb der Registerkarte *Evaluate!* der Alice-Steuerung dargestellt. Die Zuordnung eines Vorgangs kann durch Markieren eines Listeneintrags kontrolliert werden.

Wird ein Listenelement durch Anklicken ausgewählt, wird das entsprechende Bauteil blau markiert (Stütze und Listeneintrag in Abbildung 6.8) und die zugeordneten Vorgänge im rechten Teil der Tabellenansicht dargestellt.

Die im Beispiel gelb markierten Stützen besitzen, bezogen auf die als Resultat der definierten Vorgänge übernommenen Bauteilbeschreibung, abweichende Materialeigenschaften (Stahlbeton) und weisen eine Ähnlichkeit von $sim_o = 0,59$ auf. Dem Anwender wird dadurch vermittelt, dass für diese Bauteile entweder ein Vorgang zu definieren ist, oder aber die Möglichkeit besteht, die Definition eines entsprechenden Vorgangs durch Anpassung eines Vorgangs mit mittlerer Ähnlichkeit vorzunehmen.

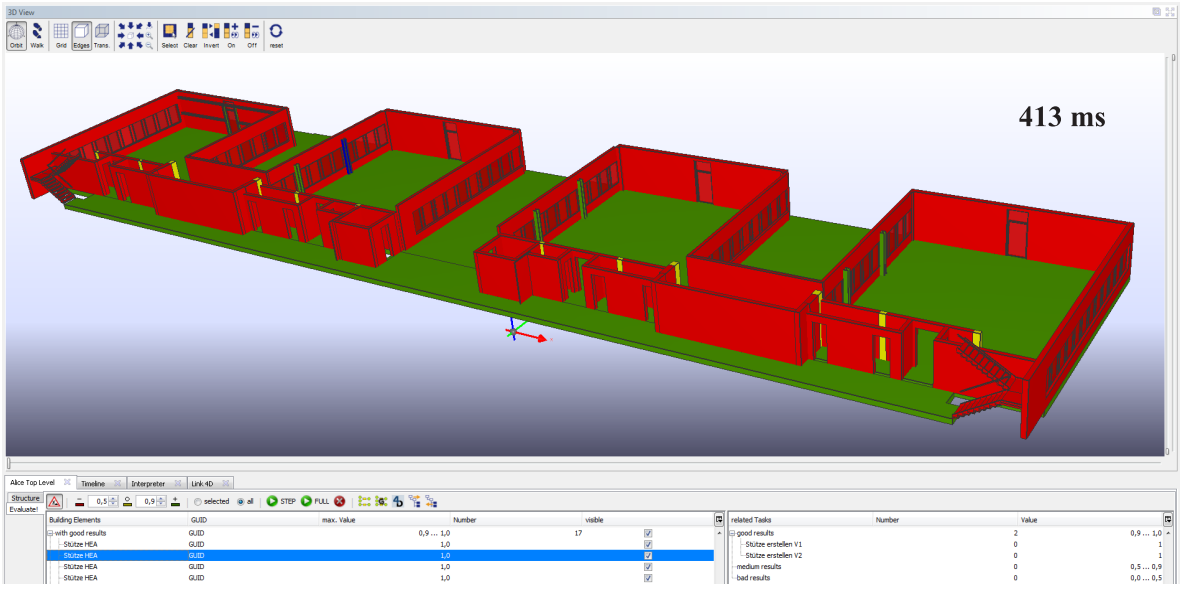


Abbildung 6.8: Ergebnis nach der Zuordnungsphase; selektiertes Bauteil der Listenansicht - Ergebnis nach 413 ms

Abbildung 6.9 zeigt die Definition eines weiteren Vorgangs zum Erstellen einer Fassadenwand. Dieser benötigt für dessen Ausführung zwei Stahlstützen als Voraussetzung, an denen die Fassadenwand vorgehangen wird.

Nach dem Einfügen des Vorgangs in die Datenbasis und erneuter Ausführung der Zuordnungsphase ergibt sich das in Abbildung 6.10 dargestellte Ergebnis.

Allen Fassadenwänden, die der Bauteilbeschreibung des Resultats des im vorherigen Schritt definierten Vorgangs entsprechen, wird dieser Vorgang zugeordnet, was durch deren grüne Markierung ersichtlich ist.

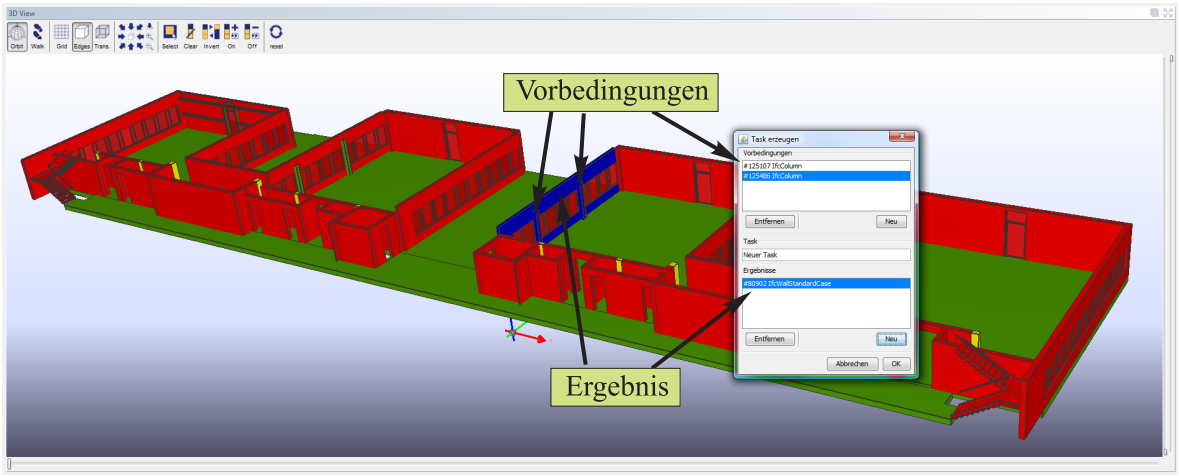


Abbildung 6.9: Definition eines Vorgangs für die Erstellung einer Fassadenwand

Weitere Wände werden gelb markiert und weisen Ähnlichkeitsmaße sim_o zwischen 0,54 und 0,59 auf. Diese Ergebnisse sind sowohl auf unterschiedliche Materialien der Bauteile als auch auf abweichende Bounding-Boxen zurückzuführen.

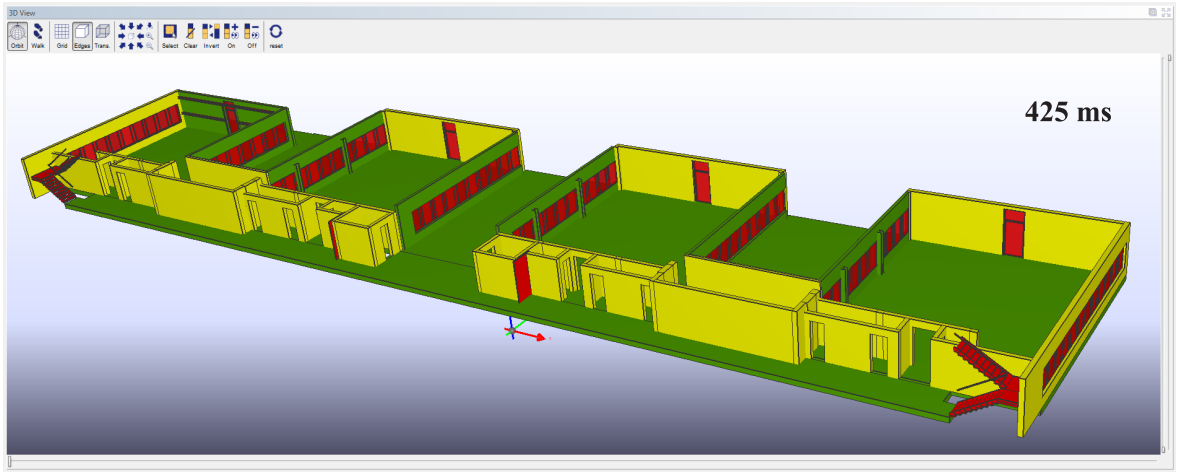


Abbildung 6.10: Ergebnis nach der Zuordnungsphase unter Berücksichtigung des Vorgangs zum Erstellen einer Fassadenwand - Ergebnis nach 425 ms

6.3.2 Beschränkung der Vorgänge für die Generierungsphase

Aus Gründen der Übersichtlichkeit wird für die Darstellung der nachfolgenden Phasen die Struktureinheit weiter untergliedert. Die entsprechenden Bauteile sind in Abbildung 6.11 blau markiert und entsprechen einer Untermenge von Bauteilen, für die ein Vorgang zugeordnet werden kann. Die Auswahl entspricht weitestgehend dem bereits theoretisch vorgestellten Beispiel des Kapitels 4.3.2 auf Seite 81.

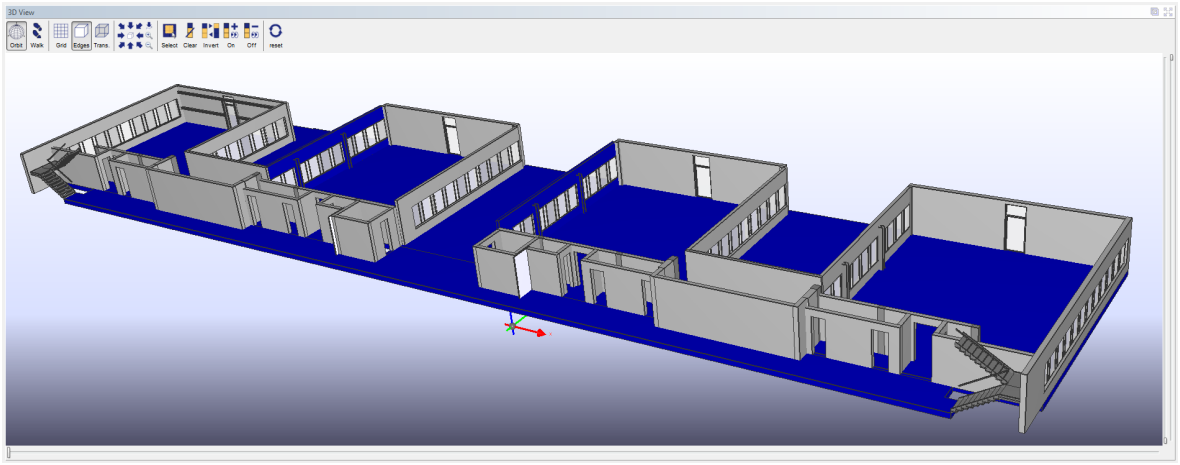


Abbildung 6.11: Auswahl der Bauteile als Struktureinheit für die Darstellung der nachfolgenden Phasen

Wird die Zuordnungsphase lediglich auf die in Abbildung 6.11 blau markierten Bauteile angewendet, ergibt sich das in Abbildung 6.12 dargestellte Ergebnis

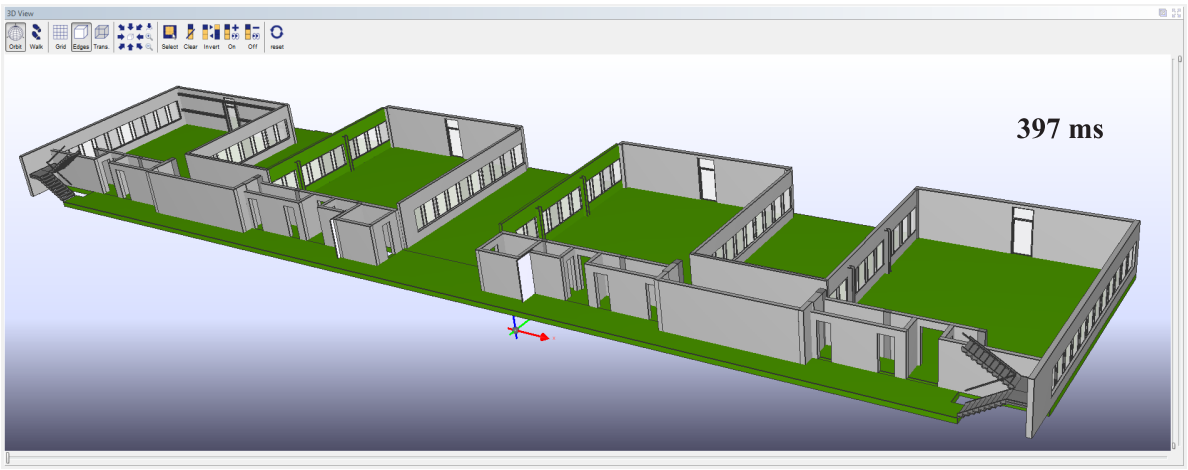


Abbildung 6.12: Ergebnis der Zuordnungsphase für die ausgewählten Bauteile - Ergebnis nach 397 ms

6.3.3 Ausführung der Generierungs- und Konfliktlösungsphase

Nach Ausführung der Zuordnungsphase für die in Kapitel 6.3.2 gewählten Bauteile ergibt sich nach Ausführung der Kopplungsphase beziehungsweise der Zusammenfassung des generierten Workflowgraphen der Struktureinheit mittels Start- und Endknoten das in Abbildung 6.13 dargestellte Ergebnis im View *AliceEvaluate* des Prototypen. Das Ergebnis nach der Verknüpfungsphase ist links, das nach der Zusammenfassung der Struktureinheit mittels Start- und Endknoten ist rechts abgebildet (siehe Kapitel 4.3.2, Seite 89 ff.).

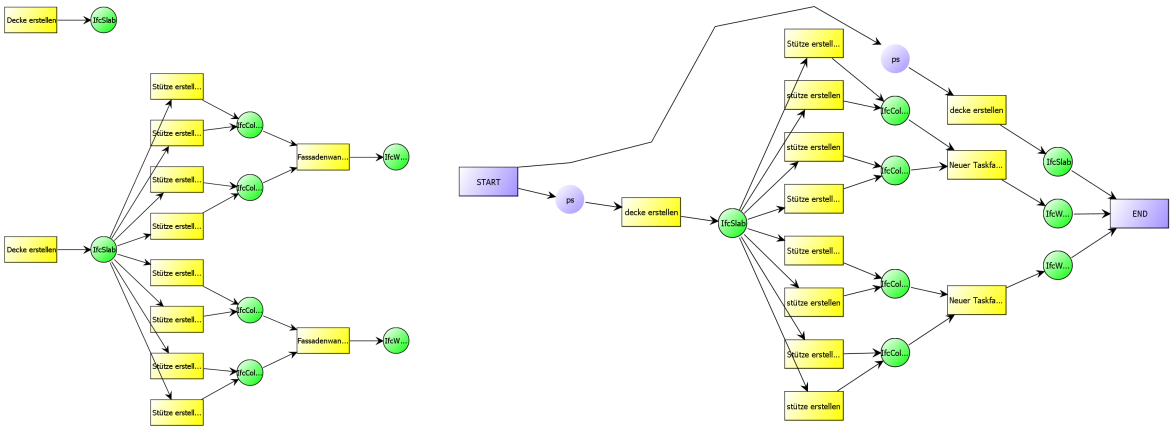


Abbildung 6.13: Darstellung des generierten Workflowgraphen im View *AliceEvaluate* nach der Verknüpfungsphase (links) und nach der Zusammenfassung des Workflowgraphen für die Struktureinheit (rechts) - Ergebnis nach 390 ms

Dabei entsprechen die in der Abbildung als gelbes Rechteck dargestellten Knoten des Workflowgraphen Vorgängen. Die als grüner Kreis dargestellten Knoten repräsentieren die mit den Vorgängen als Voraussetzung beziehungsweise als Resultat verknüpften Bauteile der betrachteten Struktureinheit. Bei violett abgebildeten Knoten handelt es sich um eingefügte Pseudoknoten, die lediglich der strukturellen Integrität des Workflowgraphen dienen.
Abbildung 6.14 zeigt das Endergebnis des generierten Workflowgraphen der Struktureinheit nach der Konfliktlösungsphase.

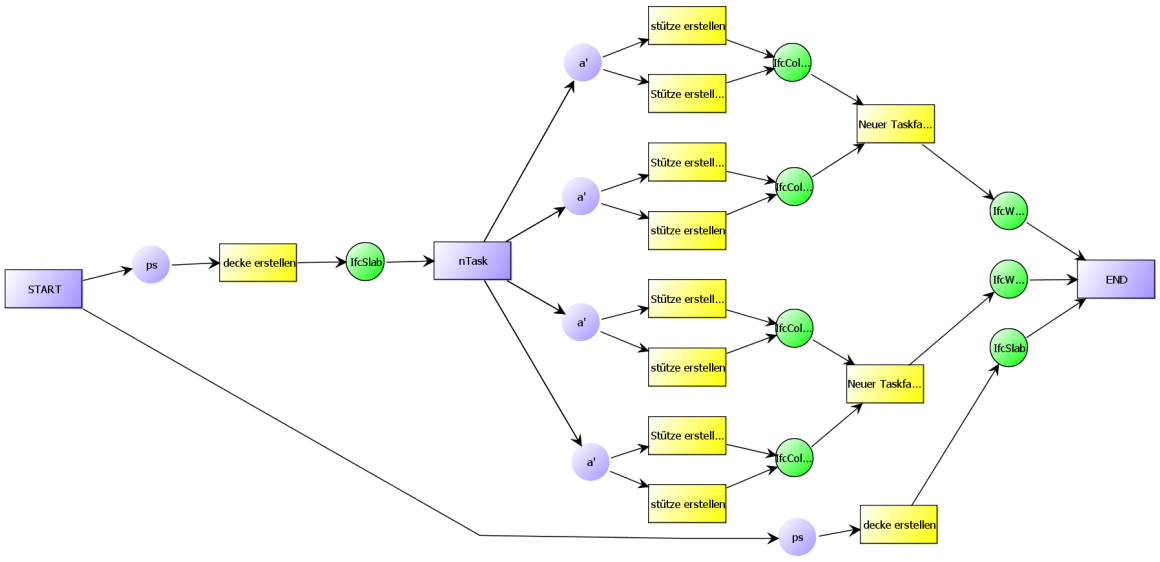


Abbildung 6.14: Darstellung des generierten Workflowgraphen im View *AliceEvalutae* nach der Konfliktlösungsphase - Ergebnis nach 157 ms

Das Endresultat entspricht dem zu erwartenden Ergebnis des in Kapitel 4.3.2 auf Sei-

te 81 theoretisch vorgestellten Beispiels. Es existiert lediglich ein zusätzlicher Vorgang für das Erstellen einer Decke, der auf die im BIM modellierte Zweiteilung der Decke zurückzuführen ist (vgl. Ergebnis des theoretischen Beispiels auf Seite 96).
Durch Selektion eines oder mehrerer Bauteilknoten des generierten Workflowgraphen besteht die Möglichkeit der Kontrolle des generierten Ergebnisses. Wird ein Knoten durch den Anwender selektiert, erfolgt die blaue Markierung des entsprechenden Bauteils im 3D-View. Abbildung 6.15 zeigt dies beispielhaft für den Ergebnisknoten des Vorgangs für das Erstellen des größeren Deckenabschnittes.

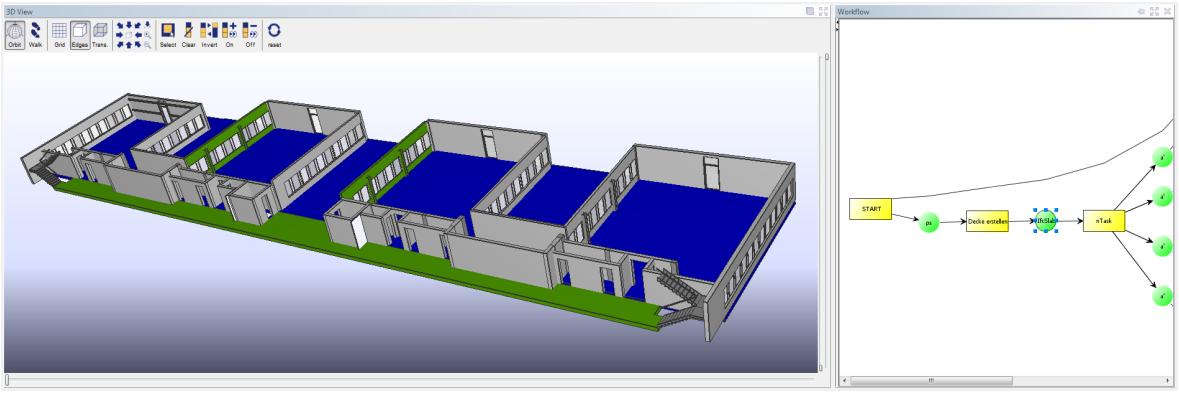


Abbildung 6.15: Durch Selektion des Ergebnisknotens des Vorgangs *Decke erstellen* blau markiertes Bauteil

6.3.4 Umsetzung der 4D-Animation

Durch die aufgrund des Modellentwurfs automatisch resultierende Verknüpfung von Bauteilen des BIM mit dem generierten Workflowgraphen ist dessen Nutzung für eine 4D-Animation problemlos realisierbar. Dies ermöglicht eine sehr gute visuelle Kontrolle des generierten Bauablaufs durch den Anwender.
Innerhalb der prototypischen Umsetzung wurde dies eingeschränkt realisiert. Durch die fehlende zeitliche Bewertung von Vorgängen wurde pauschal eine Dauer von einem Tag je Vorgang angenommen. Gleichzeitig wurde für die Art des Vorgangs vorausgesetzt, dass es sich um einen Konstruktionsvorgang handelt. Ein Vorgang erzeugt demnach das resultierende Bauteil, was innerhalb der Animation durch das Einblenden des jeweiligen Bauteils zum zeitlichen Beginn des Vorgangs dargestellt wird. Dabei wird das Bauteil während seiner Konstruktionsphase rotbraun und nach dessen Fertigstellung in der jeweiligen Materialfarbe dargestellt.

> **Anmerkung:** Die Behandlung von Varianten im Bauablauf für die 4D-Animation des Prototypen ist unberücksichtigt. Enthält der generierte Workflowgraph Varianten, werden die Vorgänge aller Varianten für die Erstellung eines Bauteils in die Animation übernommen, so dass die längste Dauer visualisiert wird. Tatsächlich müsste jedoch beim Erreichen eines Entscheidungsknotens im Workflowgraphen die Auswahl einer Variante durch den Anwender herbeigeführt werden.
> Wie bereits in Kapitel 4.4.2 diskutiert, kann eine solche Entscheidung durch Integration bekannter Methoden der Netzplantechnik in das System unterstützt werden.

Grundlage hierfür ist eine entsprechende Bewertung von Vorgängen, beispielsweise auf Basis von Zeit, Kosten oder Ressourcen.
Eine Entscheidungsunterstützung des Anwenders kann erreicht werden, indem ausgehend von einem Entscheidungsknoten alle Instanzgraphen des generierten Workflowgraphen (Siehe Kapitel 3.4.5) als mögliche Bauabläufe gegenübergestellt und gestützt auf die Ergebnisse angewendeter Methoden der Netzplantechnik vom Anwender beurteilt werden.
Infolge einer Entscheidung für eine Ablaufalternative durch den Anwender entfallen nicht gewählte Varianten für den geplanten Bauablauf. Durch eine entsprechende Markierung des nach dem Entscheidungsknoten auszuführenden Vorgangs kann die durch den Anwender als zielführend erachtete Variante auf Basis des eindeutigen Identifikators t_{guid} eines Vorgangs formal für den Bauablauf gewählt und deren Auswahl dokumentiert werden. Abgewählte Varianten bleiben jedoch innerhalb des Workflowgraphen erhalten. Sie stehen dadurch für eine alternative Entscheidung, beispielsweise auf Basis veränderter Randbedingungen, zu einem späteren Zeitpunkt weiterhin zur Verfügung.

Die Abbildungen 6.16 bis 6.19 zeigen die schrittweise Darstellung der 4D-Animation für das betrachtete Beispiel innerhalb des 3D/4D-Views des Prototypen.

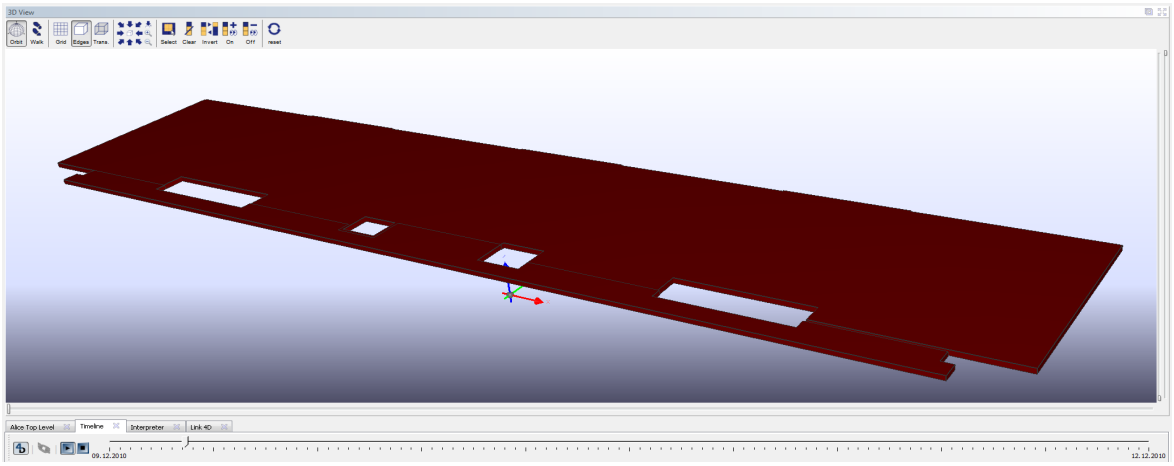


Abbildung 6.16: Darstellung der 4D-Animation - Erstellung der Deckenabschnitte

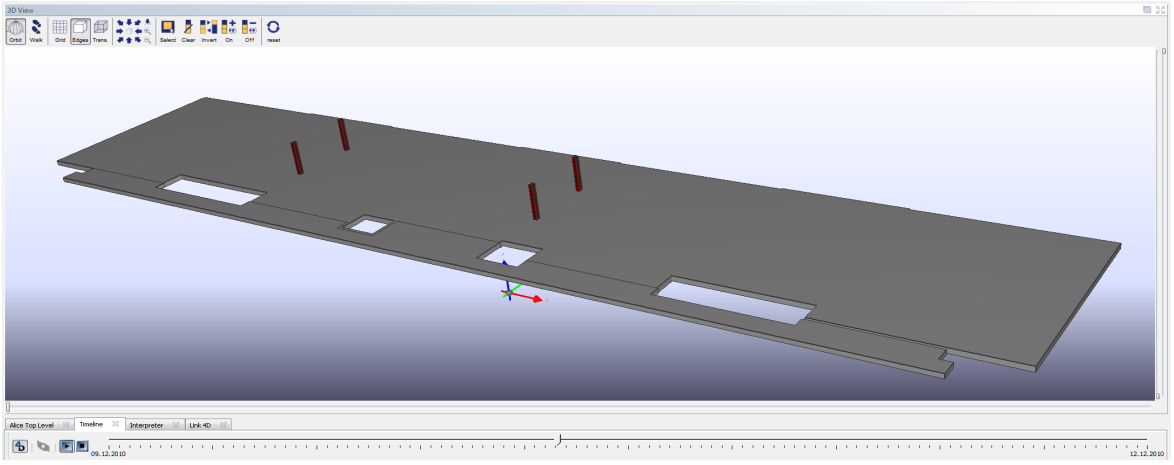


Abbildung 6.17: Darstellung der 4D-Animation - Erstellung der Stahlstützen

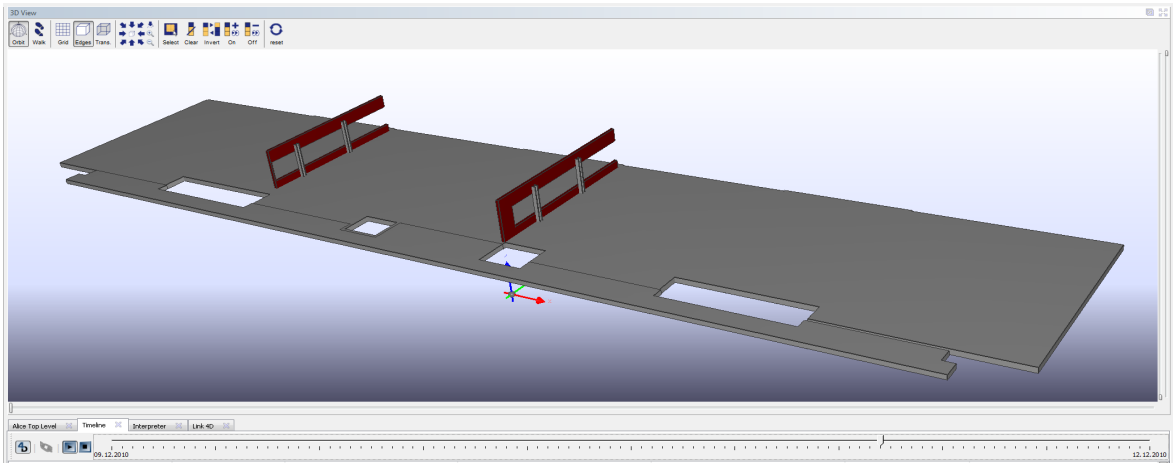


Abbildung 6.18: Darstellung der 4D-Animation - Erstellung der Fassadenwände

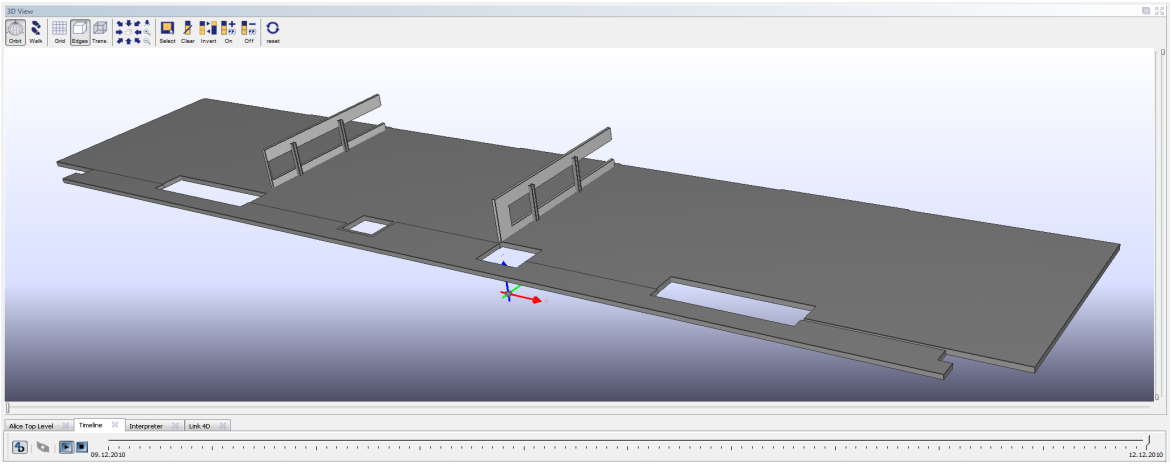


Abbildung 6.19: Endzustand der 4D-Animation

7 Zusammenfassung und Ausblick

Das vorliegende Kapitel fasst den Inhalt der Arbeit in Kurzform zusammen. Die erreichten Ergebnisse werden dabei kritisch hinterfragt und diskutiert. Abschließend werden mögliche Ansatzpunkte für eine Weiterentwicklung der vorgestellten Lösung gegeben.

7.1 Zusammenfassung

Ziel dieser Arbeit war die Entwicklung eines neuen Modells für die Unterstützung der Bauablaufplanung. Dafür wurde der unter Verwendung herkömmlicher Modelle größtenteils manuelle Vorgang der Planung betrachtet und die Vorgehensweise des Planers analysiert. Die aufgrund der dargestellten Analyse erkannten Probleme wurden aufgegriffen und unter Betrachtung aktueller Forschungsarbeiten, Bauwerksinformationsmodellen sowie existierender Softwaresysteme ein entsprechender Lösungsansatz erarbeitet. Dabei wurden im Speziellen die nachfolgenden Punkte als Ziele der Arbeit formuliert:

Varianten in der Bauablaufplanung: Durch die Integration von Varianten für Teilleistungen des Bauablaufs in den Bauablaufplan soll zum einen die transparente Dokumentation von Entscheidungen für eine bestimmte Ausführungsvariante dokumentiert werden können. Zum anderen soll das Vorhalten von Varianten eine schnelle Reaktionen bei Änderungen der Bauwerksplanung als auch bei Störungen im Bauablauf ermöglichen.

Teilautomatisierung des Planungsvorgangs: Der größtenteils manuelle Vorgang der Bauablaufplanung soll auf Basis des zu entwickelnden Modells weitestgehend automatisiert werden. Infolge der angestrebten Automatisierung werden im Wesentlichen zwei Ergebnisse verfolgt. Zum einen ist durch die Integration von Varianten ein erhöhter Planungsaufwand zu erwarten. Um der dadurch drohenden Egalisierung der genannten Vorteile zu begegnen, muss der resultierende Aufwand für den Planer deutlich reduziert werden. Zum anderen wird durch einen verringerten Planungsaufwand eine verbesserte Nachführung des Bauablaufplans infolge von Änderungen in der Bauwerksplanung erwartet.

Wiederverwendbarkeit von Bauablaufplanungen: Geplante und bereits ausgeführte Bauabläufe stellen eine nicht zu unterschätzende Wissensbasis für zukünftig zu planende Bauabläufe dar. Dieses Wissen soll für den Planer verfügbar gemacht

werden. Durch eine entsprechend ausgerichtete Abbildung des Modells soll ein hoher Grad an Wiederverwendbarkeit während der Planung definierter Vorgänge und Bauabläufe erreicht werden.

Integration einer 4D-Animation in den Prozess der Bauablaufplanung: Durch die Integration einer 4D-Animation in den Prozess der Bauablaufplanung soll eine hohes Maß an Kontrolle des geplanten Ablaufs erreicht werden. Infolge einer direkten Kopplung des Bauablaufplans mit Bauteilen eines BIM soll ebenfalls auf die Vollständigkeit des Bauablaufplans geschlossen werden können.

Ausgehend vom Lösungsansatz wurde untersucht, ob die für den Planungsvorgang notwendigen Informationen aus digitalen BIM extrahiert und die Vorgehensweise der Bauablaufplanung auf deren Basis formal abgebildet werden kann.
Aufbauend auf den daraus gewonnenen Erkenntnissen wurde ein Modell entwickelt, das eine teilautomatisierte Arbeitsweise für die Bauablaufplanung mit Ausrichtung auf das Erreichen der genannten Ziele erlaubt. Das Modell wurde formal auf Basis der Mengen- und Relationenalgebra beschrieben. Notwendige Algorithmen wurden entwickelt und mittels Pseudocode dargestellt und erläutert.
Für den Tragfähigkeitsnachweis der vorgestellten Lösung wurde basierend auf dem entwickelten Modell die softwaretechnische Umsetzung geplant und prototypisch realisiert. Die Anwendbarkeit wurde exemplarisch an einem Beispiel gezeigt.

7.2 Diskussion der Ergebnisse

Das vorgestellte Konzept beabsichtigt, den Aufwand für die Erstellung von Bauablaufplänen zu verringern und gleichzeitig die Qualität der Planung zu verbessern. Dabei werden Varianten für das Erreichen von Teilleistungen im Bauablauf berücksichtigt und bleiben innerhalb des geplanten Bauablaufs enthalten. Als notwendige Informationsbasis wird das Vorhalten des geplanten Bauwerks als BIM vorausgesetzt. Im Ergebnis ist ein den Planer unterstützendes Werkzeug entstanden, das die teilautomatisierte Generierung von Bauablaufplänen erlaubt:

Automatisierung: Verschiedene Prozesse der Bauablaufplanung konnten formal abgebildet und infolgedessen automatisiert werden. Zu diesen Prozessen gehört die Identifikation notwendiger Vorgänge des Bauablaufs sowie die Bestimmung von deren Reihenfolge für die Erstellung von Bauabschnitten bzw. Struktureinheiten. Die Reihenfolge der Realisierung von Bauabschnitten kann im gesamten Bauprozess jedoch nicht automatisch bestimmt werden. Hierfür wurde die herkömmliche Arbeitsweise zugrunde gelegt und in die dreidimensionale, modellbasierte Arbeitsweise übertragen. Das vorgestellte Konzept stützt sich dabei auf die Arbeit von [Tulke 2010] und ermöglicht die Definition von Arbeitszonen sowie die Definition von deren Reihenfolge im Bauablauf durch den Anwender. Diese Vorgehensweise entspricht im weitesten Sinne der herkömmlichen Grobterminplanung.

Varianten im Bauablauf: Die Berücksichtigung von Varianten im Bauablauf konnte auf Basis des vorgestellten Modells erreicht werden. Steht mehr als ein Vorgang innerhalb der Datenbasis für die Erstellung eines Bauteils zur Verfügung, führt dies innerhalb der Generierungsphase automatisch zur Integration dieser Varianten in den Bauablauf.

Anmerkung: Die nach dem Generierungsprozess im Bauablauf enthaltenen Varianten sind jedoch mitunter aus technologischer Sicht nicht frei miteinander kombinierbar. Beispielsweise kann eine Ausführungsvariante, die infolge des Einsatzes von technischem Gerät die maximale Tragkraft einer Geschossdecke übersteigt, nicht ausgeführt werden, obwohl dies ohne Probleme in einem ähnlichen Projekt möglich wäre.
Wird angenommen, dass alle Vorgänge der Datenbasis korrekt definiert wurden, führt der Generierungsprozess jedoch ausschließlich zu richtigen Ergebnissen. Dennoch kann die Existenz nicht kombinierbarer Varianten nicht ausgeschlossen werden. Eine entsprechende Behandlung dieser Problematik wurde nicht eingehend betrachtet. Nachfolgender Lösungsvorschlag wird jedoch als zielführend angesehen:

Unverträgliche Variantenkombinationen können nach der Entscheidung für eine bestimmte Ablaufvariante anhand des eindeutigen Identifikators t_{guid} eines Vorgangs ausgeschlossen werden. Durch Referenzierung dieses Identifikators können Bedingungen definiert werden, die eine Kombination von Aufgaben innerhalb eines Weges im generierten Graphen ausschließen. Wird eine Entscheidung für eine Variante innerhalb des Bauablaufs getroffen, können alle nachfolgenden Instanzgraphen auf Basis solcher Bedingungen geprüft werden. Unverträgliche Varianten können zu diesem Zeitpunkt markiert und vom Bauablauf ausgeschlossen werden.

4D-Integration: Insbesondere die Integration von 4D-Funktionalität in den Prozess der Bauablaufplanung, die sich aufgrund der gewählten Art der Modellierung ergibt, stellt eine neue Qualität im Vergleich zu herkömmlichen Modellen und Methoden dar. Infolge der daraus resultierenden visuellen Kontrolle des generierten Bauablaufs können fehlende Vorgänge bereits innerhalb der Planungsphase besser identifiziert und Probleme frühzeitig erkannt und beseitigt werden. Die nachträgliche Erstellung von Modellen für eine 4D-Animation bzw. Simulation entfällt beziehungsweise kann aufgrund der bestehenden Verknüpfungen zwischen Bauteilen des BIM und dem generierten Bauablauf gewonnen werden. In diesem Zusammenhang ist anzumerken, dass die prototypische Umsetzung die Auswahl einer Variante beim Erreichen eines Entscheidungsknotens nicht unterstützt, da nur die prinzipielle Tauglichkeit des Modells in dieser Hinsicht im Fokus der Arbeit stand. Diese konnte auf Basis der prototypischen Umsetzungen nachgewiesen werden. Eine diesbezügliche Erweiterung wird als unproblematisch eingeschätzt.

Wiederverwendbarkeit: Durch die vorgeschlagene Art und Weise der Modellierung von Vorgängen wird ein hohes Maß an Wiederverwendbarkeit von Bauablaufplanungen erreicht. Vom Anwender definierte Vorgänge werden innerhalb einer Datenbasis vorgehalten und stehen zukünftig für die automatische Zuordnung zu Bauteilen zur Verfügung. Dies führt zu einer Reduzierung notwendiger Neudefinitionen von Vorgängen. Gleichzeitig spiegelt diese ständig wachsende Datenbasis das Wissen der das System einsetzenden Anwender wider. An dieser Stelle ist eine Qualitätssteigerung der Bauablaufplanung möglich, da auch weniger erfahrene Planer auf dieses Wissen Zugriff haben. In gleichem Maß birgt es die Gefahr von Fehlplanungen, falls Vorgänge nicht korrekt definiert und in die Datenbasis eingetragen werden.
Das erreichbare Ergebnis der Bauablaufplanung unter Verwendung des entwickelten Modells ist in starker Abhängigkeit zur Datenqualität des zugrunde liegenden BIM zu sehen. Die für die Zuordnung von Vorgängen betrachteten Bauteilinformationen, wie beispielsweise unter Verwendung der IFC dargestellt, können zum großen Teil eindeutig aufgelöst werden (Bauteiltyp und Bounding-Box). Einzig die Materialangaben eines Bauteils, die den größten Einfluss auf die Wahl eines passenden Vorgangs haben, erfolgen in der Regel auf der Basis von Text[1]. Dies erschwert den formalen Vergleich von Bauteilbeschreibungen bei Ermittlung des Ähnlichkeitsmaßes, da diese Angaben in der Regel durch den Nutzer im entsprechenden CAD-System eingepflegt werden. Der Grund hierfür sind beispielsweise fehlende Materialangaben im BIM, Schreibfehler oder die Verwendung unterschiedlicher Bezeichnungen desselben Materials aufgrund landestypischer Bezeichnungen. Dieser Umstand senkt entsprechend den Grad der Wiederverwendbarkeit von Vorgängen der Datenbasis.

Anwendbarkeit: Infolge der Entkopplung von Vorgängen setzt das entwickelte Modell eine modellbasierte Arbeitsweise voraus, die in einigen Punkten deutlich von der derzeitigen Vorgehensweise in der Bauablaufplanung abweicht. Für das in Kapitel 6 dargestellte Beispiel kann, beginnend mit einer leeren Datenbasis, durch die Definition von drei Vorgängen bereits 21,6% der betrachteten Bauteile ein Vorgang zugeordnet werden[2]. Ein mit dem System vertrauter Anwender benötigt für die Definition der gezeigten Vorgänge ca. 2 Minuten. Wenngleich sich die benötigte Zeit infolge der diskutierten Bewertung von Vorgängen erhöht, ist bedingt durch die automatische Zuordnung von Vorgängen eine für den Anwender kompfortable Arbeitsweise erkennbar. Für die Definition von Arbeitszonen und Achsen ist eine weiterführende Unterstützung des Anwenders, wie beispielsweise deren automatische Erzeugung anhand eines Rasters und die Vorgabe einer Arbeitsrichtung, denkbar.
Weiterhin trägt die Möglichkeit einer visuellen Kontrolle in allen Phasen der Generierung zu einer höheren Akzeptanz des generierten Bauablaufs bei.

[1] auf Basis der IFC ist auch die Angabe physikalischer Größen möglich
[2] 24 von 111 Bauteilen wurde ein Vorgang zugeordnet

Übertragbarkeit auf weitere Anwendungsgebiete: Die Anwendung des vorgestellten Verfahrens wurde für die Planung des Bauablaufs gezeigt. Die prinzipielle Vorgehensweise und der Einsatz des vorgestellten Modells ist nicht darauf beschränkt, sondern auch in anderen Anwendungsbereichen möglich. Beispielsweise sind vielfältige Anwendungen im Bereich des Gebäudemanagements denkbar. So ist unter anderem eine automatisierte Zuordnung von Wartungs- oder Prüfvorgängen für die technische Gebäudeausrüstung oder auch die Zuordnung von Vorgängen für die Pflege und Reinigung von Bodenbelägen möglich.

Abschließende Bemerkungen: Das vorgestellte Modell basiert auf der Definition, dass Vorgänge als Resultat genau ein Bauteil erzeugen. Die Ausprägung eines Bauteils ist dabei vom zugrunde liegenden BIM abhängig. In der Regel werden Bauteile atomar abgebildet. Eine Zusammenfassung von mehreren Bauteilen zu einem Gesamtresultat während der Bauausführung ist jedoch denkbar. Beispielsweise kann die Erstellung einer Wand mit einem darin befindlichen Fenster aus zwei Vorgängen bestehen. Dies sind im Einzelnen das Erstellen der Wand und der anschließende Einbau des Fensters. Alternativ könnte die Wand in Form eines Fertigteils mit bereits integriertem Fenster eingebaut werden. Dieser Umstand wurde im Modell berücksichtigt, insofern die Modellierung der Bauteilbeschreibungen für Resultate und Voraussetzungen generisch umgesetzt wurde. Prinzipiell ist die Modellierung von zusammengesetzten Bauteilen dadurch möglich und die vorgestellten Algorithmen weiterhin anwendbar.

Für die Gruppierung von Bauteilen des BIM zu einem Resultat wird für die Ermittlung weiterer Vorgänge folgende Vorgehensweise vorgeschlagen:

Nach erfolgter Zuordnungsphase des vorgestellten Modells werden, ausgehend vom Startknoten einer Struktureinheit, alle möglichen Gruppierungen resultierender Bauteile im jeweiligen Instanzgraphen gebildet. Kann für eine daraus resultierende Bauteilgruppe ein Vorgang innerhalb der Datenbasis identifiziert werden, so handelt es sich um eine Variante für die Erstellung dieser Bauteilgruppe. Der Entscheidungsknoten einer solchen Variante kann über die direkten Vorgänger des Vorgangs bestimmt werden, dessen resultierender Bauteilknoten den kleinsten Vorgängergrad innerhalb der Bauteilgruppe aufweist. Der dazu korrespondierende XOR-Join kann nach dem Bauteilknoten der Gruppe mit dem größten Vorgängergrad angeschlossen werden. Dieser referenziert dann entsprechend dem Modell alle Bauteile dieser Gruppe.

Die genannte Methodik basiert dabei auf der Annahme, dass Bauteile einer Gruppierung entsprechend der in dieser Arbeit vorgestellten Art der Modellierung voneinander abhängig sind.

7.3 Ausblick

Im Hinblick auf die Überführung des vorgestellten Modells in die praktische Anwendung wurde ein DFG[3]-Transferprojekt beantragt. Innerhalb des beantragten Projektes soll in enger Zusammenarbeit mit dem Kooperationspartner OBERMEYER Project Management GmbH Leipzig die prinzipielle praktische Anwendbarkeit des vorgestellten Modells untersucht und gegebenenfalls erweitert beziehungsweise angepasst werden. Gegenstand der Kooperation ist die Konzeption modellbasierter Methoden zur Spezifikation von Vorgangsbedingungen für die automatische Berechnung von Terminplänen unter Einsatz des implementierten Prototypen durch den Kooperationspartner.

Einige der im Team entwickelten Komponenten des Prototypen werden beziehungsweise wurden unter der Internetdomain www.openifctools.de veröffentlicht (siehe auch [Theiler u. a. 2009], [Theiler 2010] und [Theiler u. a. 2010]. Dies betrifft insbesondere den IFC STEP PARSER/WRITER sowie weitere Komponenten des 3D/4D-Viewers. Dieser wurde inzwischen sowohl Forschungs- als auch Studienprojekten zur Verfügung gestellt und findet somit bereits Verwendung in Forschung (siehe [Katranuschkov u. a. 2010] bzw. http://www.mefisto-bau.de[4]) und Lehre (siehe [Kohls u. a. 2010]). Komponenten des 3D-Viewers wurden im Rahmen einer Masterarbeit (siehe [Theiler 2011]) am Lehrstuhl Informatik im Bauwesen der Bauhaus-Universität Weimar weiterentwickelt und in den IFC-Zertifizierungsserver GTDS[5] der buildingSMART Alliance integriert.

Die Ergebnisse der Arbeit wurden bereits in Kapitel 7.2 vorgestellt und diskutiert. Infolge der Diskussion wurden bereits verschiedene Aspekte hinsichtlich einer möglichen Erweiterung des Modells vorgeschlagen. Weiterhin werden nachfolgende Punkte für eine Weiterentwicklung des vorgestellten Modells als aussichtsreich beziehungsweise zielführend angesehen:

Bewertung von Vorgängen: Die Bewertung von Vorgängen wurde bereits innerhalb des dieser Arbeit assoziierten Forschungsprojekts hinsichtlich einer Risikoeinschätzung unter Verwendung der Methoden TOPSIS[6] [Chen u. Hwang 1992] und AHP[7] [Saaty 1990] prototypisch umgesetzt (vgl. [Mikulakova 2007] und [Mikulakova u. König 2009]). Der Ausbau des Modells bezüglich einer Bewertung von Vorgängen hinsichtlich Zeit, Kosten und Ressourcen erscheint für eine optimierte Auswahl von Ablaufvarianten zweckdienlich. Gleichzeitig wird die 4D-Darstellung des Bauablaufs durch die daraus resultierende zeitliche Bewertung entscheidend aufgewertet.

Typisierung von Vorgängen: Für die Planung verschiedener Bauvorhaben sollte die Datenbasis hinsichtlich einer möglichen Zuordnung von Vorgängen zu verschie-

[3]Deutsche Forschungsgemeinschaft
[4]Abruf: 27.01.2011
[5]Global Testing Documentation Server; http://gtds.buildingsmart.com - Abruf: 27.01.2011
[6]Technique for Order Preference by Similarity to Ideal Solution
[7]Analytic Hierachy Process

denen Kategorien erweitert werden. Im derzeitigen Modell werden ausschließlich Vorgänge für die Erstellung von Bauteilen betrachtet. In der Praxis existiert jedoch beispielsweise auch die Möglichkeit des Rückbaus von Bauteilen. Wird ein solches Bauvorhaben unter Verwendung des derzeitigen Modells geplant, ist dies zwar möglich, den vom Rückbau betroffenen Bauteilen würden jedoch, abhängig von Inhalt der Datenbasis, auch Varianten für deren Erstellung zugeordnet. Eine entsprechende Kategorisierung der Datenbasis hinsichtlich der Art eines Vorgangs würde die gezielte Anwendung der Zuordnungsphase für bestimmte Anwendungsfälle ermöglichen, indem dieser eine Vorgangskategorie der Datenbasis zugeordnet wird. Für das genannte Beispiel des Rückbaus ist an dieser Stelle auch eine verfeinerte 4D-Darstellung zu erwarten, da von der Art des Vorgangs auf eine dem Sinn entsprechende Visualisierung geschlossen werden kann. So können beispielsweise erstellte Bauteile eingeblendet und vom Rückbau betroffene ausgeblendet werden.

Definition von Vorgängen: Für die Modellierung von Vorgängen ist in Erweiterung des vorangegangenen Punktes eine Kategorisierung der Datenbasis ebenfalls wünschenswert. Infolge einer Klassifizierung von Vorgängen auf Basis deren Zugehörigkeit von Gewerken und Ausbaustufen, wie dem Rohbau oder Innenausbau, ist eine verbesserte Planung unter Verwendung des vorgestellten Modells erreichbar. Dies ermöglicht beispielsweise die Definition eines Vorgangs „Anstrich“ der Kategorie Innenausbau, dessen Resultat sich nur auf die äußere Materialschicht eines Bauteils bezieht. Zwar unterstützt das vorgestellte Modell Bauteilbeschreibungen für Resultate und Voraussetzungen von Vorgängen mit beliebigem Detaillierungsgrad, eine zeitliche Einordnung von Vorgängen bezogen auf deren Ausbaustufe ist jedoch bisher nicht möglich. Die Zuordnung einer Vorgangskategorie der Datenbasis zu einer Struktureinheit würde die Definition verschiedener Baustufen eines Bauteils innerhalb des Bauablaufs ermöglichen.

Listings

Liste der Algorithmen

Literaturverzeichnis

[GSA 2008] U.S. General Services Administration (GSA): Statement of intention to support Building Information Models with open standards. Version: 2008. `http://www.gsa.gov/graphics/pbs/Statement_of_Intention-BIM_FINAL.pdf`. 2008. – Forschungsbericht. – Abruf: 08.11.2011

[Aalami u. a. 1998] Aalami, Florian B. ; Fischer, Martin A. ; Kunz, John C.: AEC 4D Production Model: Definition and Automated Generation / CIFE, Stanford University. 1998. – Forschungsbericht

[van der Aalst 1998] Aalst, Wil van d.: The Application of Petri Nets to Workflow Management. In: *The Journal of Circuits, Systems and Computers* 8 (1998), S. 21–66

[Aamodt u. Plaza 1994] Aamodt, A. ; Plaza, E.: Case-Based Reasoning - Foundational Issues, Methodological Variations, and System Approaches. In: *AI Communications. IOS Press* 7 (1994), Nr. 1, S. 39–59

[Apache 2010] Apache: *The Apache DB Project Documentation*. `http://db.apache.org/derby/manuals/index.html`. Version: 2010. – Abruf: 26.07.2010

[Autodesk-Consulting 2007] Autodesk-Consulting: *BIM and Project Planning*. White Paper. Autodesk, 2007. `images.autodesk.com/adsk/files/bim_project_planning_feb07_1_.pdf`. – Abruf: 28.01.2011

[Bauer 2007] Bauer, Hermann: *Baubetrieb*. 3. Auflage. Springer Verlag Berlin, Heidelberg, 2007. – ISBN: 978-3540321132

[Beetz u. a. 2010] Beetz, Jakob ; Berlo, Léon van ; Laat, Ruben de ; Helm, Pim van d.: BIMSERVER.ORG AN OPEN SOURCE IFC MODEL SERVER. In: *Proceedings of the CIB W78 2010: 27th*, 2010

[Besner u. Hobbs 2008] Besner, Claude ; Hobbs, Brian: Project Management Practice, Generic or Contextual: A Reality Check. In: *Project Management Journal* 39 (2008), March, Nr. 1, S. 16–33. – Published online in Wiley InterScience (www.interscience.wiley.com) DOI: 10, 1002/pmj 20033

[Böhme 1993] Böhme, Gert: Berlin, Heidelberg : Springer Verlag, 1993. – ISBN: 978-3540566588

[Björk 1995] Björk, Bo-Christer: *Requirements and information structures for building product data models*. Vuorimiehentie 5, Espoo, P.O.Box 42 : Technical Research Centre of Finland (VTT), 1995. – ISBN: 951-38-4783-7

[buildingSMART 2008] BUILDINGSMART: *IFC Roadmap Summary.* online. `http://www.iai-tech.org/projects/ifc-roadmap/summary`. Version: 2008. – Abruf: 08.02.2011

[buildingSmart 2010] BUILDINGSMART: *International Alliance for Interoperability, IFC2x3 TC1 documentation.* `http://www.iai-tech.org/ifc/IFC2x3/TC1/html/index.htm`. Version: 2010. – Abruf: 11.11.2010

[Chang 2009] CHANG, Han-Shuo: *Color Schemes on 4D Models in 4D Construction Management Tools*, National Taiwan University, Diplomarbeit, 2009

[Chen u. a. 2005] CHEN, Po-Han ; CUI, Lu ; WAN, Caiyun ; YANG, Qizhen ; TING, Seng K. ; TIONG, Robert L.: Implementation of IFC-based web server for collaborative building design between architects and structural engineers. In: *Automation in Construction* 14 (2005), Januar, Nr. 1, 115–128. `http://www.sciencedirect.com/science/article/B6V20-4DN173Y-1/2/8babd7674d8c7bc1069cc861c7c34c36`. – ISSN 0926–5805

[Chen u. Hwang 1992] CHEN, S.-J. ; HWANG, C.-L.: *Fuzzy Multiple Attribute Decision Making: Methods and Applications (Lecture Notes in Economics and Mathematical Systems).* Berlin : Springer-Verlag, 1992. – ISBN: 978-0-387549-98-9

[Cherneff u. a. 1991] CHERNEFF, Jonathan ; LOGCHER, Robert ; SRIRAM, D.: Integrating CAD with Construction-Schedule Generation. In: *J. Comp. in Civ. Engrg.* 5 (1991), Januar, Nr. 1, 64–84. `http://link.aip.org/link/?QCP/5/64/1`

[Claus u. Schwill 2003] CLAUS, V. ; SCHWILL, A.: *Duden. Informatik. Ein Fachlexikon für Studium und Praxis.* 1. Auflage. Mannheim : Bibliographisches Institut, Mannheim, 2003. – ISBN: 978-3411100231

[Copeland 2007] COPELAND, Tom ; JOYCE, Elizabeth (Hrsg.): *Generating Parsers with JavaCC.* Alexandria, VA 22304, USA : Centennial Books, 2007. – ISBN: 0-9762214-3-8

[Darwiche u. a. 1988] DARWICHE, Adnan ; LEVITT, Raymond E. ; HAYES-ROTH, Barbara: OARPLAN: Generating project plans by reasoning about objects, actions and resources. In: *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing* 2 (1988), S. 169–181

[Dawood u. a. 2005] DAWOOD ; SCOTT ; SRIPRASERT ; MALLASI: The virtual construction site (VIRCON) tools: An industrial evaluation. In: *ITCON* 10 (2005), S. 43–54

[Dawood u. a. 2003] DAWOOD, Nashwan ; SRIPRASERT, Eknarin ; MALLASI, Zaki ; HOBBS, Brian: Development of an integrated information resource base for 4D/VR construction processes simulation. In: *Automation in Construction* 12 (2003), Nr. 12, S. 123–131

[DIN 2009] DIN: *DIN 69900, Projektmanagement: Netzplantechnik - Beschreibungen und Begriffe; DIN 69901-1, Projektmanagementsysteme - Teil 1: Grundlagen; DIN 69901-2, Projektmanagementsysteme - Teil 2: Prozesse, Prozessmodell; DIN 69901-3, Projektmanagementsysteme - Teil 3: Methoden; DIN 69901-4, Projektmanagementsysteme - Teil 4: Daten, Datenmodell; DIN 69901-5, Projektmanagementsysteme - Teil 5: Begriffe*. 2009

[Domschke u. Drexl 2007] DOMSCHKE, Wolfgang ; DREXL, Andreas: *Einführung in Operations Research*. Berlin : Springer Verlag Berlin, 2007 (7. Auflage). – ISBN: 978-3540709480

[Dyllong u. a. 1999] DYLLONG, Eva ; LUTHER, Wolfram ; OTTEN, Werner: An Accurate Distance-Calculation Algorithm for Convex Polyhedra. In: *Reliable Computing* 5 (1999), 241-253. http://dx.doi.org/10.1023/A:1009924204609. – ISSN 1385–3139. – 10.1023/A:1009924204609

[Dzeng u. Tommelein 1997] DZENG, R. ; TOMMELEIN, I.D.: Boiler Erection Scheduling Using Product Models and Case-Based Reasoning. In: *Journal of Construction Engineering and Management* 123 (1997), Nr. 3, 338-347. http://scitation.aip.org/getpdf/servlet/GetPDFServlet?filetype=pdf&id=JCEMD4000123000003000338000001&idtype=cvips&prog=normal

[Dzeng 2000] DZENG, Ren-Jye: Development of activity network modules for expressway projects / Taiwan Expressway Engineering Bureau. Taiwan, 2000. – Technical Report

[Dzeng u. Tommelein 2004] DZENG, Ren-Jye ; TOMMELEIN, Iris D.: Product modeling to support case-based construction planning and scheduling. In: *Automation in Construction* 13 (2004), October, S. 341–360

[Dzeng u. Whang 2003] DZENG, Ren-Jye ; WHANG, Wei-Chih: Automatic schedule integration for highway projects. In: *Automation in Construction* 12 (2003), S. 447–461

[Eastman 1999] EASTMAN, C. M.: *Building Product Models: Computer Environments Supporting Design and Construction*. Boca Raton, Florida USA : CRC Press, 1999 (0849302595)

[Eastman u. a. 2008] EASTMAN, Chuck ; TEICHOLZ, Paul ; SACKS, Rafael ; LISTON, Kathleen: *BIM Handbook - A Guide to Building Infromation Modeling for Owners, Managers, Designers, Engineers, and Contractors*. John Wiley & Sons, Inc., 2008 (9780470185285)

[Enge 2005] ENGE, F.: Zustandsmodellierung als Grundlage für Ausführungsterminpläne. In: *Forum Bauinformatik*. Cottbus : Schley, F. ;Weber, L., 2005, S. 123–130. – ISBN: 3-934934-11-0

[Enge u. Huhnt 2008] ENGE, F. ; HUHNT, W.: Optimal Component Types for the Design of Construction Processes. In: *Proceedings of the XIIth International Conference on Computing in Civil and Building Engineering.* Beijing, 2008

[Enge 2010] ENGE, Felix ; BARJENBRUCH, Matthias (Hrsg.) ; GEISSLER, Karsten (Hrsg.) ; HINKELMANN, Reinhard (Hrsg.) ; HUHNT, Wolfgang (Hrsg.) ; KOCHENDÖRFER, Bernd (Hrsg.) ; PETRYNA, Yuri (Hrsg.) ; SAVIDIS, Stavros (Hrsg.) ; SCHLAICH, Mike (Hrsg.) ; SCHMID, Volker (Hrsg.) ; VOGDT, Frank U. (Hrsg.): *Muster in Prozessen der Bauablaufplanung - Ein Branch - and - Bound Verfahren zur Mustererkennung in Planungs- und Ausführungsprozessen.* Achen : Shaker Verlag, 2010. – ISBN: 978-3-8322-9053-5

[Ester u. a. 2008] ESTER, A. ; FITZNER, J. ; HOLZHEU, A. ; KOSCHORREK: Modellbasierte Terminplanung. In: *Bachelorprojekt SS 2008, Bauhaus-Universität Weimar.* Weimar : Betriebswirtschaft im Bauwesen, 2008

[Freundt 2005] FREUNDT, Martin: *Ein flexibles Modell für die Bauablaufplanung auf Basis von Graphentheorie und Fuzzy-Zahlen.* Aachen : Shaker Verlag, 2005. – ISBN: 3-8322-3533-7

[Freundt u. Beucke 2004] FREUNDT, Martin ; BEUCKE, Karl: A flexible model for scheduling building processes based on graph theory and fuzzy numbers. In: *X-th International Conference on Computing in Civil and Building Engineering ICCCBE-X.* Weimar, 2004

[Richter von Hagen u. Stucky 2004] HAGEN, Cornelia Richter v. ; STUCKY, Wolffried: *Business-Process- und Workflow-Management: Prozessverbesserung durch Prozess-Management.* Vieweg+Teubner Verlag, 2004. – ISBN: 978-3519004912

[Hamming 1950] HAMMING, R. W.: Error Detecting and Error Correcting Codes. In: *Bell System Techn. Journal* 26 (1950), Nr. 2, S. 147–160

[Hanff 2003] HANFF, Jochen: *Abhängigkeiten zwischen Objekten in ingenieurwissenschaftlichen Anwendungen.* Aachen : Shaker Verlag, 2003. – ISBN: 978-3832222802

[Hauschild 2003] HAUSCHILD, Thomas: *Computer Supported Cooperative Work-Applikationen in der Bauwerksplanung auf der Basis einer integrierten Bauwerksmodellverwaltung.* Weimar, Bauhaus-Universität Weimar, Dissertation, 2003

[Heesom u. Mahdjoubi 2004] HEESOM, D. ; MAHDJOUBI, L.: Trends of 4D CAD applications for construction planning. In: *Construction Management and Economics* 22 (2004), Nr. February, 22, S. 171–182

[Heinrich u. Huhnt 2003] HEINRICH, T. ; HUHNT, W.: Determination of Effects of Modifications during Planning Processes. In: *16. Internationalen Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen - IKM.* Bauhaus-Universität Weimar, 2003

[Hendrickson u. a. 1987] HENDRICKSON, C. ; MARTINELLI, D. ; REHAK, D.: HIERARCHICAL RULE-BASED ACTIVITY DURATION ESTIMATION. In: *Journal of Construction Engineering and Management* 113 (1987), Nr. 2, S. 288–301. – ISSN: 0733-9364

[Howard u. Björk 2008] HOWARD, Rob ; BJÖRK, Bo-Christer: Building information modelling - Experts' views on standardisation and industry deployment. In: *Advanced Engineering Informatics* 22 (2008), Nr. 2, 271–280. http://www.sciencedirect.com/science/article/B6X1X-4R8M4CW-3/2/527cf952c5d4979d15900245e1c7c7b6. – ISSN 1474–0346

[Hubbard 1990] HUBBARD, P.M.: *Constructive Solid Geometry for Triangulated Polyhedra*. Bd. CS-90-07. Department of Computer Science, Brown University, Providence, Rhode Island 02912, 1990

[Huhnt 2005] HUHNT, W.: Generating Sequences of Construction Tasks. In: *Proceedings of 22nd of W78 Conference on Information Technology in Construction*. Dresden, 2005

[Huhnt 2009] HUHNT, W.: Process Modelling in Civil Engineering. In: *Structural Engineering International* 19 (2009), February, Nr. 1, S. 91–101

[Huhnt u. Enge 2006] HUHNT, W. ; ENGE, F.: Can algorithms support the specification of construction schedules? In: *ITcon* 11 (2006), July, S. 547–564

[Huhnt 2010] HUHNT, Wolfgang: Special issue on construction informatics. In: *Advanced Engineering Informatics* 24 (2010), November, Nr. 4, 377–378. http://www.sciencedirect.com/science/article/B6X1X-512DT01-1/2/09c1d3a5aea5bc325de507649be569db. – ISSN 1474–0346

[ISO 2002a] ISO: *ISO 10303 -21: Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure*. 2002

[ISO 2002b] ISO: *ISO 10303-28: Industrial automation systems and integrationProduct data representation and exchange - Part 28: Implementation methods: XML representations of EXPRESS schema and data*. 2002

[ISO 2004] ISO: *ISO 10303-11: Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual*. 2004

[Jan Reinhardt 2004] JAN REINHARDT, James Garrett Burcu A.: SiDaCoS: Product and Process Models on Construction Sites. 2004. – Forschungsbericht

[java.net 2010] JAVA.NET: *The Java3D Project Page*. https://java3d.dev.java.net/. Version: 2010. – Abruf: 26.07.2010

[JogAmp 2010] JOGAMP: *JOGL Project Overview*. http://jogamp.org/jogl/www/. Version: 2010. – Abruf: 26.07.2010

[Jones 2010] JONES, Stephen A. (Hrsg.): *SmartMarket Report, The Business Value of BIM in Europe*. USA, Badford, MA 01730 : McGraw-Hill Construction, Research & Analytics, 2010. – ISBN: 978-1-934926-27-7

[Kang u. a. 2005] KANG, Leen-Seok ; PAULSON, Boyd C. ; HAK KIM, Joong-Min K.: Business Breakdown Structure for Construction Management and Web-based Application System. In: *ITCON* 10 (2005), S. 169–191

[Katranuschkov u. a. 2010] KATRANUSCHKOV, Peter ; WEISE, Matthias ; WINDISCH, Ronny ; FUCHS, Sebastian ; SCHERER, Raimar J.: BIM-BASED GENERATION OF MULTI-MODEL VIEWS. In: *Proceedings of the CIB W78 2010: 27th International Conference*. Cairo, Egypt, 2010

[Kawachi u. Suzuki 2000] KAWACHI, Katsuaki ; SUZUKI, Hiromasa: Distance Computation between Non-convex Polyhedra at Short Range Based on Discrete Voronoi Regions. In: *Geometric Modeling and Processing 2000. Theory and Applications*, 2000, S. 123–128

[Kähkönen u. Leinonen 2003] KÄHKÖNEN, Kalle ; LEINONEN, Jarkko: Visual product chronology as a solution for accessing building product model data. In: *CIB w78 Conference*, 2003

[Klinger 2003] KLINGER, Axel: Strukturanalyse von Workflow-Graphen. In: KAPKE, Kai (Hrsg.) ; WULF, Alexander (Hrsg.): *Forum Bauinformatik - Junge Wissenschaftler forschen*. Aachen : Shaker Verlag, 2003, S. 147–159

[König 2004] KÖNIG, M.: *Ein Prozessmodell für die kooperative Gebäudeplanung*. Aachen : Shaker Verlag, 2004

[König u. Beißert 2009] KÖNIG, M. ; BEISSERT, U.: Construction Scheduling Optimization by Simulated Annealing. In: CALDAS, C. H. (Hrsg.) ; O'BRIEN, W. J. (Hrsg.) ; CHI, Seokho (Hrsg.) ; GONG, Jie (Hrsg.) ; LUO, Xiaowei (Hrsg.): *Proceedings of the 26th Annual International Symposium on Automation and Robotics in Construction*. Austin, Texas, United States : The University of Texas at Austin, June 2009, S. 183–190. – ISBN: 978-0-578-02312-0

[Kochendörfer u. a. 2010] KOCHENDÖRFER, Bernd ; LIEBCHEN, Jens H. ; VIERING, Markus G.: *Bau-Projekt-Management: Grundlagen und Vorgehensweisen*. Bd. 4. Auflage. Vieweg+Teubner, 2010. – ISBN: 978-3834804969

[Kohls u. a. 2010] KOHLS, K. ; BLOCK, M. ; MARX, A. ; HAMM, M.: BIM-basierte Bauablaufsimulation von Hochbauten. In: *22. Forum Bauinformatik*. Berlin : Technische Universität Berlin, Fachgebiet Bauinformatik, 2010

[Kolodner 1993] KOLODNER, Janet: *Case-Based Reasoning (Morgan Kaufmann Series in Representation & Reasoning)*. San Mateo : Morgan Kaufman Publishers, 1993 http://opac.ub.uni-weimar.de/DB=1/SET=5/TTL=1/CMD?ACT=SRCHA&IKT=1016&SRT=YOP&TRM=+Kolodner. – ISBN: 978-1558602373

[Koo u. Fischer 2000] KOO, B. ; FISCHER, M.: Feasibility Study of 4D CAD in Commercial Construction. In: *Journal of Construction Engineering and Management* 126 (2000), Nr. 4, S. 251–260

[Koo u. Fischer 1998] KOO, Bonsang ; FISCHER, Martin: Feasibility Study of 4D CAD in Commercial Construction. In: *CIFE - Technical Report*. O CIFE, Civil and Environmental Engineering Dept., Stanford University, Terman Engineering Center, Mail Code: 4020, Stanford, CA 94305-4020 : CIFE - CENTER FOR INTEGRATED FACILITY ENGINEERING, Stanford University, 1998 (118)

[Koo u. Fischer 2003] KOO, Bonsang ; FISCHER, Martin: Formalizing Construction Sequencing Constraints for Rapid Generation of Schedule Alternatives / CIFE, Stanford University. Version: 2003. http://cife.stanford.edu/online.publications/WP075.pdf. CIFE, 2003. – Forschungsbericht

[Koschorrek u. a. 2008] KOSCHORREK, S. ; ESTER, A. ; TAUSCHER, E. ; TULKE, J.: IFC-basiertes Modell für die 4D-Bauablaufanimation. In: WINDISCH, G. R.and F. R.and Faschingbauer (Hrsg.) ; KATRANUSCHKOV, P. (Hrsg.): *Forum Bauinformatik 2008*. Dresden : Institut für Bauinformatik Dresden, September 2008. – ISBN: 987-3-86780-090-7

[Laidlaw u. a. 1986] LAIDLAW, D.H. ; TRUMBORE, W.B. ; J.F., Hughes: Constructive Solid Geometry for Polyhedral Objects. In: *Proceedings of SIGGRAPH 86* Bd. 2. New York, USA : ACM, 1986

[Leen-Seok u. a. 2004] LEEN-SEOK, K. ; YONG-SU, L. ; JUNG-MIN, K.: 4D-Modellsystem; Funktionsanalyse für die Bauplanung. In: *Computer Spezial -Software für Architekten Ingenieure Bauunternehmen* Feb. (2004), S. 13–15

[Liebich 2009] LIEBICH, Thomas: *IFC 2x Edition 3 Model Implementation Guide*. Version 2.0. http://www.iai-tech.org/downloads/accompanying-documents/guidelines/IFC2xModelModeling Support Group, 2009. – Abruf: 11.11.2010

[Liebich u. Gersching 2010] LIEBICH, Thomas ; GERSCHING, Erwin: IT Unterstützung für die Wertschöpfungskette Bau. In: *buildingSMART Strategiekonzept* (2010). http://www.buildingsmart.de/pdf/openBIM_Strategiekonzept.pdf. – Abruf: 19.01.2011

[Mikulakova 2007] MIKULAKOVA, E.: Bewertung von Ausführungsalternativen zur Entscheidungsfindung. In: MERKEL, A. P. (Hrsg.) ; SCHÜTZ, R. (Hrsg.) ; WIESSFLECKER, T. (Hrsg.): *Forum Bauinformatik 2007*, Verlag der Technischen Universität Graz, September 2007, S. 135–143. – ISBN: 987-3-902465-86-3

[Mikulakova u. König 2009] MIKULAKOVA, E. ; KÖNIG, M.: Multi-criteria Evaluation of Construction Processes. In: *Computing in Engineering EG-ICE Conference 2009*. Berlin, Germany : Shaker Verlag, 2009, S. 210–217

[Mikulakova u. a. 2008] MIKULAKOVA, E. ; KÖNIG, M. ; TAUSCHER, E. ; BEUCKE, K.: Knowledge Management for Construction Scheduling. In: *IABSE ICT Conference 2008*. Helsinki : International Association for Bridge and Structural Engineering, 2008

[Mikulakova 2010] MIKULAKOVA, Eva: *Wissensbasierte Bauablaufplanung mit Fallbasiertem Schließen*, Bauhaus-Universität Weimar, Fakultät Bauingenieurwesen, eingereichte Dissertation, 2010

[Mikulakova u. a. 2010] MIKULAKOVA, Eva ; KÖNIG, Markus ; TAUSCHER, Eike ; BEUCKE, Karl: Knowledge-based schedule generation and evaluation. In: *Advanced Engineering Informatics* (2010)

[Möller 1997] MÖLLER, T.: A Fast Triangle-Triangle Intersection Test. In: *Journal of Graphics Tools 2 (2)* (1997), S. 25–30

[Morad u. Beliveau 1994] MORAD, A. A. ; BELIVEAU, Y. J.: Geometric-Based Reasoning System for Project Planning. In: *Journal of Computing in Civil Engineering* 8 (1994), S. 52–71

[Navinchandra u. a. 1988] NAVINCHANDRA, D. ; SRIRAM, D. ; LOGCHER, R. D.: Ghost: Project Network Generator. In: *J. Comp. in Civ. Engrg.* 2 (1988), Juli, Nr. 3, 239–254. `http://link.aip.org/link/?QCP/2/239/1`

[Oracle 2010a] ORACLE: *Java DB Reference*. `http://developers.sun.com/javadb/reference/index.jsp`. Version: 2010. – Abruf: 26.07.2010

[Oracle 2010b] ORACLE: *Java Platform, Standard Edition 6 API Specification*. `http://download.oracle.com/docs/cd/E17409_01/javase/6/docs/api/`. Version: 2010. – Abruf: 22.07.2010

[Oracle 2010c] ORACLE: *The Source for Java Developers*. `http://java.sun.com/`. Version: 2010. – Abruf: 16.07.2010

[Pahl u. Damrath 2000] PAHL, Peter J. ; DAMRATH, Rudolf: *Mathematische Grundlagen der Ingenieurinformatik*. Springer-Verlag Berlin, 2000

[Plume u. Mitchell 2007] PLUME, Jim ; MITCHELL, John: Collaborative design using a shared IFC building model–Learning from experience. In: *Automation in Construction* 16 (2007), Januar, Nr. 1, 28–36. `http://www.sciencedirect.com/science/article/B6V20-4HNYMGD-4/2/5cdbee78b977373b0e1dd3e26e5606f3`. – ISSN 0926–5805

[Porkka u. Kähkönen 2007] PORKKA, Janne ; KÄHKÖNEN, Kalle: Software development approaches and challenges of 4D product models. In: *24th CIB w78 Conference*, 2007 (CIB W78 Conference)

[Pries 2007] PRIES, August: IFC als Standardformat des Building Information Modells (BIM) - Planungs- und baubegleitende Datenerfassung / CAD-Stelle Bayern. Version: 2007. `http://www.buildingsmart.de/pdf/ps_2007-02.pdf`. 2007. – Forschungsbericht

[Richter 2008] RICHTER, M. M.: *Studies in Computational Intelligence*. Bd. 73. Springer Verlag Berlin, 2008. – 25–90 S.

[Robert 2008] ROBERT, Christian R.: *Entwicklung eines Bauzeitennachtrages und Untersuchung der Möglichkeit der verbesserten Durchsetzbarkeit durch den Einsatz von 4D-Visualisierungswerkzeugen*, Fachhoschule Kaiserslautern, Diplomarbeit, 2008

[Saaty 1990] SAATY, T. L.: How to Make a Decision: The Analytic Hierarchy Process. In: *European Journal of Operational Research* 48 (1990), Nr. 1, S. 9–26

[Scheid 2000] SCHEID, Harald: *Duden Rechnen und Mathematik; Das Lexikon für Schule und Praxis*. 6. Auflage. Dudenverlag Mannheim, Leipzig, Wien, Zürich, 2000

[Sheppard 2004] SHEPPARD, Laurel M.: Virtual Building for Construction Projects. In: *IEEE Computer Graphics and Applications* (2004), Nr. January/February, S. 6–12

[Staub-French u. a. 2008] STAUB-FRENCH, Sheryl ; RUSSELL, Alan ; TRAN, Ngoc: Linear Scheduling and 4D Visualisation. In: *Journal of Computing in Civil Engineering* 22 (2008), S. 192–205

[Tantisevi u. Akinci 2009] TANTISEVI, Kevin ; AKINCI, Burcu: Transformation of a 4D product and process model to generate motion of mobile cranes. In: *Automation in Construction* 18 (2009), October, S. 458–468

[Tanyer u. Aouad 2005] TANYER, Ali M. ; AOUAD, Ghassan: Moving beyond the fourth dimension with an IFC-based single project database. In: *Automation in Construction* 14 (2005), S. 15–32

[Tauscher u. a. 2007] TAUSCHER, Eike ; MIKULAKOVA, Eva ; KÖNIG, Markus ; BEUCKE, Karl: Generating Construction Schedules with Case-Based Reasoning Support. In: SOIBELMAN, Lucio (Hrsg.) ; AKINCI, Burcu (Hrsg.): *Proceedings of the 2007 International Workshop on Computing in Civil Engineering*. Reston, Virginia : American Society of Covil Engineers, 2007, S. 119–126

[Theiler u. a. 2009] THEILER, M. ; TAUSCHER, E. ; TULKE, J. ; RIEDEL, T.: Visualisierung von IFC-Objekten mittels Java3D. In: BOTH, P. (Hrsg.) ; KOCH, V. (Hrsg.): *21. Forum Bauinformatik 2009*, Universitätsverlag Karlsruhe, September 2009, S. 149–159

[Theiler u. a. 2010] THEILER, M. ; TAUSCHER, E. ; TULKE, J. ; RIEDEL, T.: Boolesche Operationen für die Visualisierung von IFC-Gebäudemodellen. In: *22. Forum Bauinformatik*. Berlin : Technische Universität Berlin, Fachgebiet Bauinformatik, 2010

[Theiler 2010] THEILER, Michael: *Documentation and conceptual development of software components for the execution of geometric Boolean set operations on the basis of Java3D,*. Informatik im Bauwesen, Bauhaus-Universität Weimar, Studienarbeit, May 2010

[Theiler 2011] THEILER, Michael: *Interaktive Visualisierung von Qualitätsdefiziten komplexer Bauwerksinformationsmodelle auf Basis der Industry Foundation Classes (IFC) in einer webbasierten Umgebung*. Informaik im Bauwesen Weimar, Bauhaus-Universität Weimar, Masterarbeit, Januar 2011

[Tulke 2010] TULKE, Jan: *Kollaborative Terminplanung auf Basis von Bauwerksinformationsmodellen*. Verlag der Bauhaus-Universität Weimar, 2010 (Schriftenreihe Informatik in Architektur und Bauwesen). – ISBN: 978-3-86068-416-0

[Tversky 1977] TVERSKY, A.: Features of Similarity. In: *Psychological Review* 84 (1977), Nr. 2, S. 327–352

[Uher 2003] UHER, Ghomas E.: *Programming and Scheduling Techniques*. UNSW Press, 2003 (0868407259)

[de Vries u. Hanrik 2007] VRIES, Bauke de ; HANRIK, Jeroen M. J.: Generation of a construction planning from a 3D CAD model. In: *Automation in Construction* 16 (2007), S. 13–18

[Waly u. Thabet 2002] WALY, Amed F. ; THABET, Walid Y.: A Virtual Construction Environment for preconstruction planning. In: *Automation in Construction* 12 (2002), S. 139–154

[Watson 2010] WATSON, Alastair: BIM - a driver for change. In: TIZANI, W. (Hrsg.): *Proceedings of the International Conference on Computing inCivil and Building Engineering*. Nottingham, UK : Nottingham University Press, June 2010. – ISBN: 978-1-907284-60-1

[Wender 2009] WENDER, Katrin: *Das virtuelle Bauwerk als Informationsumgebung für die Planung im Bestand: Zur Organisation und Strukturierung einer digitalen Bauwerksakte*. Weimar : Verlag der Bauhaus-Universität Weimar, 2009. – ISBN: 978-3-86068-393-4

[Willenbacher 2002] WILLENBACHER, Heiko: *Interaktive verknüpfungsbasierte Bauwerksmodellierung als Integrationsplattform für den Bauwerkslebenszyklus*. Weimar, Bauhaus-Universität Weimar, Dissertation, 2002

[Winch u. Kelsey 2005] WINCH, G.M. ; KELSEY, J.: What do construction project planners do? In: *International Journal of Project Management* 23 (2005), Nr. 2, S. 141–149

[Zhou u. a. 2009] ZHOU, Wei ; HEESOM, David ; GEORGAKIS, Panagiotis ; NWAGBOSO, Cristopher ; FENG, Adam: An interactive approach to collaborative 4D construction planning. In: *ITCON* 14 (2009), S. 30–47

[Zimmermann u. a. 2010] ZIMMERMANN, Jürgen ; STARK, Christoph ; RIECK, Julia: *Projektplanung: Modelle, Methoden, Management*. Berlin : Springer Verlag Berlin, 2010 (2. Auflage). – ISBN: 978-3642118784

A Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Bei der Auswahl und Auswertung von Material haben mir andere Personen weder entgeltlich noch unentgeltlich geholfen.

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Ich versichere ehrenwörtlich, dass ich nach bestem Wissen die reine Wahrheit gesagt und nichts verschwiegen habe.

Weimar, den 04. Januar 2011 Eike Tauscher

B Über den Autor

B.1 Lebenslauf

Persönliche Daten

Name:	Eike Tauscher
Geburtstag und -ort:	30. November 1974 in Stollberg/Erzgeb.
Familienstand:	ledig

Ausbildung

09/1981 – 06/1991:	Polytechnische Oberschule Neuwürschnitz
09/1991 – 06/1993:	Technisches Gymnasium Oelsnitz Erzgeb./Auerbach, Abschluss: Allgemeine Hochschulreife
07/1993 – 06/1994:	Grundwehrdienst
09/1994 – 07/2003:	Studium des Bauingenieurwesens an der Bauhaus-Universität Weimar, Studienrichtung: Bauinformatik, Abschluss: Diplom-Ingenieur (Dipl.-Ing.)

Beruflicher Werdegang

04/1998 – 04/1999:	Studentische Hilfskraft am Bereich *Arbeitsgruppe Wasser und Umwelt* der Bauhaus-Universität Weimar Mitarbeit am Projekt MAMBO - Multimedia im Fernstudium
04/2001 – 04/2003:	Studentische Hilfskraft am Bereich *Arbeitsgruppe Wasser und Umwelt* der Bauhaus-Universität Weimar Administration der Rechner der Arbeitsgruppe
10/2003 – 06/2004:	Forschung und Lehre am *Institute of Structural Engineering* der University of Stellenbosch in Südafrika Visualisierung mittels Java3D, Applikationen basierend auf Plugin Technologie, Seminare zum Kurs*Geometric Modelling in a modern CAD Environment*)
08/2004 – 12/2010:	Wissenschaftlicher Mitarbeiter an der Professur *Informatik im Bauwesen* (Prof. Beucke)
08/2004 – 06/2006:	Sonderforschungsbereich 524 *Werkstoffe und Konstruktionen für die Revitalisierung von Bauwerken* Projektbereich D *Informationsverarbeitung und Kommunikation* Teilbereich D1 (2. Förderperiode): *Methoden und Werkzeuge zur*

	Planung der Bauplanung
09/2006 – 08/2009	DFG-Projekt *Modellierung von Ausführungsvarianten in der Bauprozessplanung*
09/2007 – heute	Entwicklung von Werkzeugen für die Handhabung von IFC Daten *OPENIFCTOOLS*

B.2 Publikationen

[Tauscher 2005] Tauscher, Eike: Ein Modell für die Planung und Steuerung von Bauprozessen. In: Schley, F. (Hrsg.); Weber, L. (Hrsg.): *Forum Bauinformatik 2005*. Cottbus: Brandenburgische Technische Universität Cottbus, 2005. ISBN 3-934934-11-0, S. 141-148

[König u. a. 2006] König, Markus; Beucke, Karl; Tauscher, Eike: Management and Evaluation of Alternative Construction Tasks. In: *Digital Proceedings of the 11th International Conference on Computing in Civil and Building Engineering (ICCCBE-XI)*. Montreal: Université du Québec, 2006

[König u. Tauscher 2006] König, Markus; Tauscher, Eike: Berechnung von Bauabläufen mit verschiedenen Ausführungsvarianten. In: *Digital Proceedings of the 17th International Conference on the Applications of Computer Science and Mathematics in Architecture and Civil Engineering (IKM)*. Weimar: Bauhaus-Universität Weimar, 2006. ISSN 1611-4085

[Koch u. a. 2006] Koch, Christian (Hrsg.); Richter, Torsten (Hrsg.); Tauscher, Eike (Hrsg.): *Forum Bauinformatik 2006*. Verlag der Bauhaus-Universität Weimar, 2006. ISBN 978-3-86068-291-3

[Tauscher 2007a] Tauscher, Eike; Mikulakova, Eva; Beucke, Karl: Generating Construction Schedules with Case-Based Reasoning Support. In: *2007 International Workshop on Computing in Civil Engineering*. Pittsburgh, Pensylvania: American Society of Civil Engineers (ASCE), 2007. ISBN 978-0-7844-0937-4, S. 119-126

[Tauscher 2007b] Tauscher, Eike: Ein Modell für das Generieren von Bauablaufplänen mit Ausführungsvarianten auf Basis von Aufgaben mit Randbedingungen. In: Merkel, A. (Hrsg.); Schütz, R. (Hrsg.); Wiessflecker, T. (Hrsg.): *Forum Bauinformatik 2007* Graz: Verlag der Technische Universität Graz, 2007. ISBN 987-3-902465-86-3, S. 161-168

[Mikulakova u. a. 2008a] Mikulakova, Eva; König, Markus; Tauscher, Eike; Beucke, Karl: Case-Based Reasoning for Construction Tasks. In: *Proceedings of the 12th International Conference on Computing in Civil and Building Engineering (ICCCBE-XII)*. Beijing: Tsinghua University, 2008. ISBN 978-7-302-18670-0

[Mikulakova u. a. 2008b] Mikulakova, Eva; König, Markus; Tauscher, Eike; Beucke, Karl: Knowledge Management for Construction Scheduling. In: *Proceedings of the International Conference on Information and Communication Technology (ICT) - for Bridges, Buildings and Construction Practice*. Helsinki: International Association for Bridge and Sructural Engineering, 2008

[Koschorreck 2008] Koschorreck, Sabine; Ester, Alexandra; Tauscher, Eike; Tulke, Jan: IFC-basiertes Modell für die 4D-Bauablaufplanung. In: Windisch, R. (Hrsg.); Faschingbauer, G. (Hrsg.); Katranuschkov, P. (Hrsg.): *Forum*

Bauinformatik 2008. Dresden: Technische Universität Dresden, 2008. ISBN: 987-3-86780-090-7, S. 33-41

[Tauscher 2009] Tauscher, Eike; Mikulakova, Eva; Beucke, Karl; König, Markus: Automated Generation of Construction Schedules based on the IFC Object Model. In: *2009 International Workshop on Computing in Civil Engineering.* Austin, Texas: American Society of Civil Engineers (ASCE), 2009. ISBN 978-0-7844-1052-3, S. 666-675

[Theiler 2009] Theiler, Michael; Tauscher, Eike; Tulke, Jan; Riedel, Thomas: Visualisierung von IFC-Objekten mittels Java3D. In: Both, P. (Hrsg.); Koch, V. (Hrsg.): *Forum Bauinformatik 2009.* Karlsruhe: Universitätsverlag Karlsruhe, 2009. ISBN 978-3-86644-396-9, S. 149-159

[Theiler 2010] Theiler, Michael; Tauscher, Eike; Tulke, Jan; Riedel, Thomas: Boolesche Operationen für die Visualisierung von IFC-Gebäudemodellen In: Forum Bauinformatik 2010, Krämer, T. (Hrsg.); Richter, S. (Hrsg.); Enge, F. (Hrsg.); Kraft, B. (Hrsg.): *Forum Bauinformatik 2010.* Berlin: Shaker Verlag Aachen, 2010. ISBN 978-3-8322-9456-4, S. 251-258

[Mikulakova u. a. 2010] Mikulakova, Eva; König, Markus; Tauscher, Eike; Beucke, Karl: Knowledge-based schedule generation and evaluation In: *Advanced Engineering Informatics*, Nr. 24, S. 389-403, Elsevier, 2010